VDS (VoP Development System)

Users Guide

Arin George, Sabrina Hames, Hanzi Yang & Trinh Dinh
Copyright © Signalogic, Inc.
August 2002
Revision B
http://www.signalogic.com/vds.htm

TABLE OF CONTENTS

1. INTRODUCTION 6

1.1 Documentation 6
1.1.1 DirectDSP Documentation 6
1.1.2 Algorithm IDE Software 6
1.1.3 External Help Files 6

2. HARDWARE 7

2.1 SigC54xx-PC/104 board 7
2.1.1 Equipment Overview 7
2.1.2 SigC54xx-PC/104 Board Installation 7

2.2 SigC67xx-PC/104 board 8
2.2.1 Equipment Overview 8
2.2.2 SigC67xx-PC/104 Board Installation 8

2.3 SigC54xx Development Board 9
2.3.1 Equipment Overview 9
2.3.1.1 Power Circuit 9
2.3.1.2 Ac-Dc Adapter 9
2.3.1.3 ATX Supply 9
2.3.2 SigC54xx Development Board Installation 9

2.4 Application-Specific Equipment 11
2.4.1 Audio Module 11
2.4.1.1 Mictor Connector 12
2.4.1.2 Audio I/O Cable 13
2.4.1.3 Audio Breakout Cable 14
2.4.2 VDS Development System Equipment 14

3. SOFTWARE INSTALLATION 16

3.1 DirectDSP Demo Programs 16
3.1.1 DirectDSP Debugger Demo Program 16

3.2 VDS Software Installation 16

4. SOFTWARE OPERATION 17

4.1 Diagnostic Tests in VDS 20
4.1.1 Low-Level Loopback (Audio Loopback) 23
4.1.2 Algorithm Loopback 26
4.1.3 Host Loopback 27
4.1.4 Using More than One Channel 29

4.2 Closing VDS 29

5. ALGORITHM DSP SOFTWARE 30

5.1 DSP Source Code Interface 30
5.1.1 DSP Coding Example Applications 30
5.1.2 Adding Application-Specific C Code 31
5.1.3 Inserting Application-Specific C Code into the Real-Time Stream 32
5.1.3.1 UserProc: Real-Time 'C' Code Entry Point 32
5.1.3.2 UserProc Application Examples 34
5.1.4 DSP Source Code Access from Host Programs 38
5.1.4.1 DSP Source Code Access and Control from Hypersignal Software 38
5.1.4.2 DSP Source Code Access and Control from DirectDSP Programs 39
5.1.4.2.1 DirectDSP API Summary 41
5.1.4.2.2 C/C++ Examples in DirectDSP 42
5.1.4.2.3 MATLAB Examples in DirectDSP 47
5.1.4.2.4 Visual Basic Examples in DirectDSP 54
5.1.5 C54xx Source Code Interface Structure 61
5.1.5.1 C54xx Source Code Interface Interrupt Service Routines 62
5.1.5.2 C54xx Source Code Interface Host PC Communication 62
5.1.5.3 C54xx Source Code Interface Buffers and Real-Time Streaming 63
5.1.6 C54xx Source Code File List 63
5.1.7 C54xx Source Code Modification Process 79
5.1.7.1 C54xx Source Code Interface Modification Notes 81

5.2 C67xx Source Code Interface 82
5.2.1 C67xx Example Applications 82
5.2.2 Adding Application-Specific C Code 83
5.2.3 Inserting Application-Specific C Code into the Real-Time Stream 84
5.2.3.1 UserProc: Real-Time 'C' Code Entry Point 85
5.2.3.2 UserProc Application Examples 87
5.2.4 C67xx Source Code Access from Host Programs 90
5.2.4.1 C67xx Source Code Access and Control from Hypersignal Software 91
5.2.4.2 C67xx Source Code Access and Control from DirectDSP Programs 92
5.2.4.2.1 DirectDSP API Summary 93
5.2.4.2.2 C/C++ Examples in DirectDSP 94
5.2.4.2.3 MATLAB Examples in DirectDSP 100
5.2.4.2.4 Visual Basic Examples in DirectDSP 106
5.2.5 C67xx Source Code Interface Structure 113
5.2.5.1 C67xx Source Code Interface Interrupt Service Routines 114
5.2.5.2 C67xx Source Code Interface Host PC Communication 114
5.2.5.3 C67xx Source Code Interface Buffers and Real-Time Streaming 115
5.2.6 C67xx Source Code File List 115
5.2.7 C67xx Source Code Modification Process 116
5.2.7.1 C67xx Source Code Interface Modification Notes 118

5.3 Interface from Application C Code 119

5.4 MELP Source Code Software 123
5.4.1 Modifying MELP Source Code 123

5.4.1.1 MELP Source Code Modification Notes 124
5.5 MP3 Source Code Software 124
APPENDIX A C54XX SOURCE CODE LICENSE AGREEMENT 125
APPENDIX B C54XX TALKER PROGRAM 126
APPENDIX C MELP LINKER COMMAND AND MAP FILES 132
APPENDIX D C6X01 TALKER PROGRAM 156
APPENDIX E C6X11 TALKER PROGRAM 166

TABLE OF FIGURES:

Figure 2-1 SigC54x/PC104 Board Mounted on a "Passive Carrier Card" 7
Figure 2-2 SigC67x/PC104 Board Mounted on a "Passive Carrier Card" 8
Figure 2-3 Base I/O Address Settings for ISA Interface Card 10
Figure 2-4 SigC54x Development System with ISA Interface Card 10
Figure 2-5 Jumper Location for power on the SigC54xx Development Board 11
Figure 2-6 Audio Module Top View Diagram 12
Figure 2-7 Mictor Connector Pin-out Diagram 13
Figure 2-8 Audio I/O Cable Pin-out Diagram 14
Figure 4-1 VDS System Configuration Tab 17
Figure 4-2 The DSP HW Configuration Tab 18
Figure 4-3 DSP Hardware Manager Dialog Button 18
Figure 4-4 DSP Hardware Manager Dialog Box 19
Figure 4-5 Loopback Analog Properties 20
Figure 4-6 VDS System Configuration Tab 21
Figure 4-7 The DSP HW Configuration Tab 21
Figure 4-8 DSP Hardware Manager Dialog Button 22
Figure 4-9 DSP Hardware Manager Dialog Box 22
Figure 4-10 Loopback Analog Properties 23
Figure 4-11 VDS GUI with Audio Loopback Selected 24
Figure 4-12 VDS Engine Manager Status Window 24
Figure 4-13 VDS Status Window Display with Audio Loopback Enabled 25
Figure 4-14 VDS GUI with Algorithm Loopback Selected 26
Figure 4-15 VDS Status Window Display with Algorithm Loopback Enabled 27
Figure 4-16 VDS GUI with Host System Loopback Enabled 28
Figure 4-17 VDS Status Window Display with Host System Loopback Enabled 28
Figure 4-18 Loopback Mode Analog Properties 29

1. Introduction

1.1 Documentation

1.1.1 DirectDSP Documentation

For most DSP/data acquisition hardware that Signalogic supports, the off-the-shelf DirectDSP software is the interface layer between the Algorithm IDE Software, or graphical user interface, and the hardware. In addition to providing general Visual C/C++, MATLAB, Visual Basic, and LabVIEW interfaces, DirectDSP software also provides an interface layer between DSP hardware and high-level applications developed by Signalogic. Instances include DSPower-Block Diagram, Real-Time Composer and the Algorithm IDE.

For general documentation for DirectDSP software, please look in the help files included on the distribution CD or available after installation in the "Signalogic DSP Software" folder or group. Select the help icon "Reference Guide"; from table of contents select "Function Reference", then "DLL Functions", then "Hardware Library". Note that functions are presented in the documentation with C/C++ syntax, but are very similar to their counterparts when called in Visual Basic or MATLAB format. Specifically, in MATLAB, functions are most often the same except without the "DS" prefix. For MATLAB reference, the .m example files are always the best source. For Visual Basic, function names are identical, but calling parameter format can sometimes vary; please refer to the files hwlib.bas, enmgr.bas, and hwmgr.bas, located on (default installation) subdirectory \dspower\hwlib\vb.

1.1.2 Algorithm IDE Software

1.1.3 External Help Files

2. Hardware

2.1 SigC54xx-PC/104 board

2.1.1 Equipment Overview

2.1.2 SigC54xx-PC/104 Board Installation

1) Follow proper ESD precautions and make sure the host machine is turned off and is unplugged.

2) Make sure that there is an audio module and a SigC54xx DSP module inserted into the appropriate sockets on the PC104 card. Normally the SigC54xx module is inserted in the module site closest to the host connector (site labeled "Module #1" on the board)

3) Make sure the I/O base address is set to 0x320. This is done by setting the rotary hex switch on the PC/104 board to “2”.

4) Mount the SigC54xx-PC/104 board on a "Passive Carrier Card" (see Figure 2-1 below).

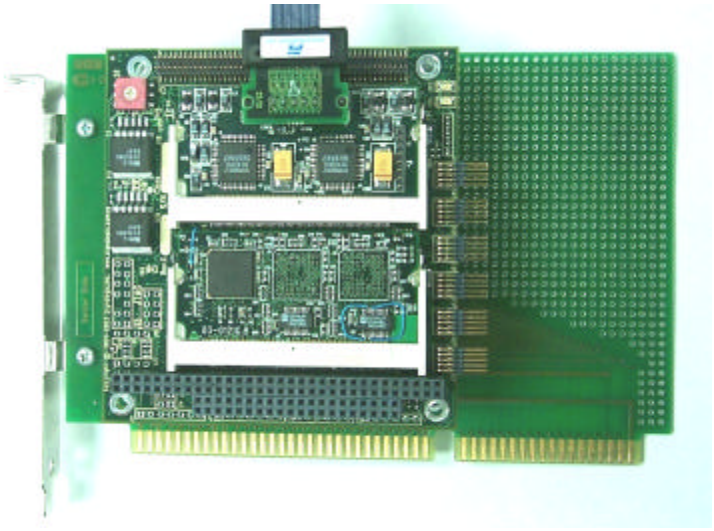


Figure 2-1 SigC54x/PC104 Board Mounted on a "Passive Carrier Card"

5) Plug the carrier card into an available 16-bit ISA slot inside the host machine.

6) Power up the host machine.

7) Wait for the computer to boot completely.

2.2 SigC67xx-PC/104 board

2.2.1 Equipment Overview

2.2.2 SigC67xx-PC/104 Board Installation

1) Follow proper ESD precautions and make sure the host machine is turned off and is unplugged.

2) Make sure that there is an audio module and a SigC67xx DSP module inserted into the appropriate sockets on the PC104 card. Normally the SigC67xx module is inserted in the module site closest to the host connector (site labeled "Module #1" on the board)

3) Make sure the I/O base address is set to 0x320. This is done by setting the rotary hex switch on the PC/104 board to “2”.

4) Mount the SigC67xx-PC/104 board on a "Passive Carrier Card" (see Figure 2-2 below).

Figure 2-2 SigC67x/PC104 Board Mounted on a "Passive Carrier Card"

5) Plug the carrier card into an available 16-bit ISA slot inside the host machine.

6) Power up the host machine.

7) Wait for the computer to boot completely.

2.3 SigC54xx Development Board

2.3.1 Equipment Overview

2.3.1.1 Power Circuit

The SigC54xx Development System board has two power circuit options: ac-dc adapter and ATX supply. These two options ***should not be connected at the same time***.

2.3.1.2 Ac-Dc Adapter

A center-positive 6v ac-dc adapter may be used for power circuit input. In this case, the power circuit will provide 5v, 3.3v, and 2.5v outputs,). Currently, if the SigC54xx Development System board is fully populated with three (3) 72-pin SODIMM modules, each with:

-three (3) 100 MHz C54xx DSP processors

-three (3) 128k x 16 SRAM devices, 8 nsec

then the default 2.5v voltage regulator used on the board is not capable enough. If this becomes a problem, please contact Signalogic for a suggested replacement component that can handle more current.

Note that a center-positive adapter is absolutely required. There is no bridge circuit on the SigC54xx Development System board; use of a center-negative adapter could cause damage to board components, and should be avoided. Note also that a 6v adapter is the maximum limit. Adapters with output voltages higher than this should not be used, as they may place excessive strain on the 3.3v and not consistent with cap V used later (or 1.8v) core-voltage regulators on the board.

2.3.1.3 ATX Supply

A standard PC-style "ATX" power supply may also be used for power circuit input. In this case, the power circuit will provide 5v, 3.3v, 2.5v, and not consistent with cap V used later. The ATX option should be used in situations where analog I/O modules are installed; typically, these modules require both not consistent with cap V used later and –5v.

2.3.2 SigC54xx Development Board Installation

1) Follow proper ESD precautions and make sure the host machine is turned off and is unplugged.

2) Make sure that there is an audio module and a SigC54xx DSP module inserted into the appropriate sockets on the development board. Normally the SigC54xx module is inserted in the module site closest to the host connector (site labeled "Module #1" on the board)

3) Make sure the I/O base address on the ISA card is set to 0x340. This is done by setting the base I/O address switch located on the edge of the ISA Card to

Ù	Ù	Ú	Ù	Ú	Ú
up	up	down	up	down	down

4) The switch setting should be read left to right, looking at edge of card as shown in Figure 2-3 below.

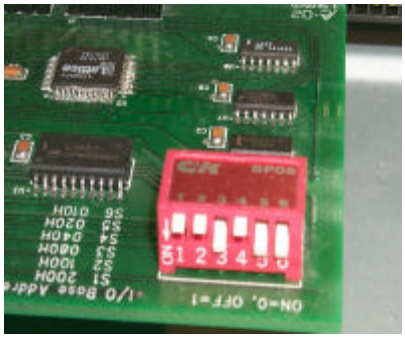


Figure 2-3 Base I/O Address Settings for ISA Interface Card

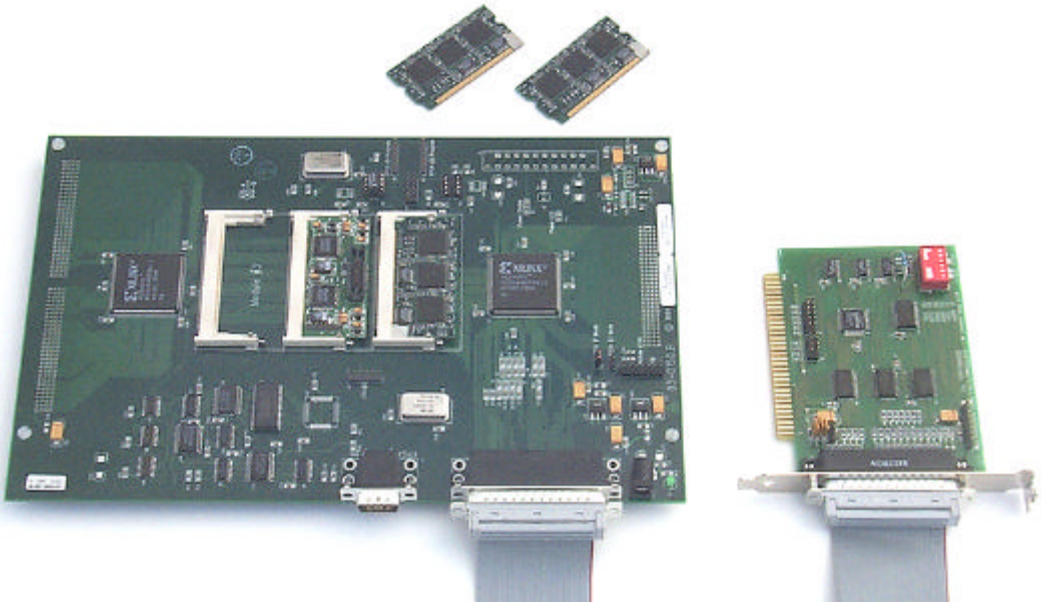


Figure 2-4 SigC54x Development System with ISA Interface Card

5) Insert the ISA interface card into an ISA slot on the host machine.

6) Connect the development board and ISA interface card via provided 37-pin ribbon cable.

7) Power up the board by placing a jumper on the first two pins on S1 jumper (Power Enable).

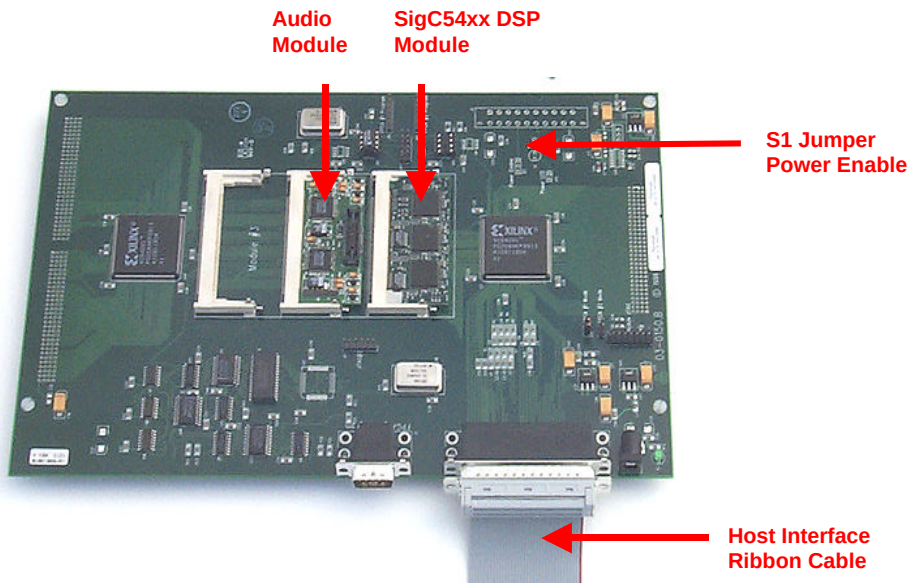


Figure 2-5 Jumper Location for power on the SigC54xx Development Board

8) Wait for the computer to boot completely.

General comment regarding processor core voltage:

SigC54xx development boards shipped from Signalogic should have the appropriate voltage set for the particular processor modules provided. If for any reason you switch out the processor module type, make sure to configure the correct core voltage as listed below:

- SigC549 – 2.5v
- SigC5409 – 1.8v
- SigC5416 – 1.5v

2.4 Application-Specific Equipment

2.4.1 Audio Module

The following installation should already be performed on deliverable VDS Development Systems and other algorithm development systems, but is documented here for completeness.

1) Audio Modules should be installed in module sites 2 or 3.

2) A SigC549 processor module used with an Audio Module needs an on-module jumper connecting TP5 (on DSP side) to edge pin 46 (on memory side). This allows the processor to use its XF (X flag) signal to reset the codecs on the Audio Module.

3) For Rev. B SigC54xx base boards, module sites that hold Audio Modules need three (3) jumpers, one for XF flag connection to module edge and two others for ±5V connection to module edge. Below is a summary of the SigC54xx Development System board jumpers:

(1) Connect +5V to Audio Module socket (typically Module #2 site) pin 23.
(2) Connect –5V to Audio Module socket (typically Module #2 site) pin 26
(3) Connect processor socket (typically Module #1 site) and Audio Module socket pin 46 together.

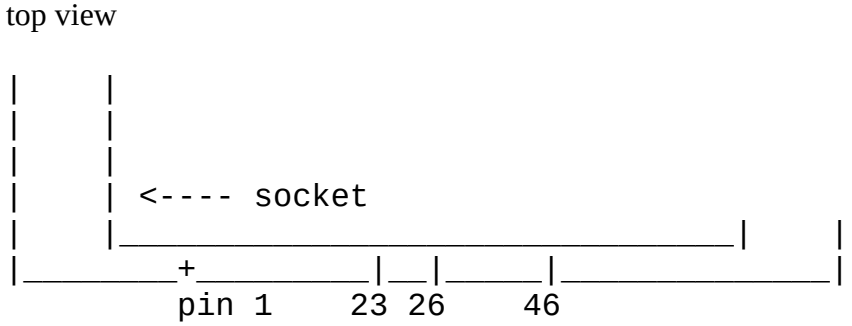


Figure 2-6 Audio Module Top View Diagram

On Rev. B SigC54xx base boards, +5V can be obtained from C51 and –5V from pin 18 of J2 (ATX power supply connector). Relatively thick jumper wires should be used as these are power connections.

Note that for Rev. C SigC54xx baseboards, no jumpers are required. However, the processor module jumper (XF signal) is still required.

IMPORTANT AUDIO MODULE NOTES

1) The small surface-mount oscillator on the Audio Module is sensitive to ESD. Please handle unconnected Audio Modules at all times with careful ESD precautions. *Always ensure that you are grounded or at least touching a computer chassis when installing or removing Audio Modules.*

2) When connecting an audio I/O cable to an Audio Modules, it is recommended to turn off the SigC54xx Development System, remove the Audio Module, connect the cable to the Mictor connector (see section 2.4.1.1 below), and re-install the Audio Module. This will prevent excessive flexing of the Audio Module card, which can eventually lead to intermittent connections due to bad solder joints on the surface-mount components on the Audio Module.

2.4.1.1 Mictor Connector

The Audio Module uses a “Mictor” style connector to provide very high density, high signal-integrity connections in the small space required by the Audio Module. Each conductor in a Mictor cable is shielded individually, and all shields are tied to a common ground using separate center posts inside the Mictor connector.

A 38-pin Mictor plug connector is used on the Audio Module, which mates to one end of the audio I/O cable (see section 2.4.1.2 below). If an Audio Module is in use, an audio I/O cable will likely be required to bring audio and digital signals out to external equipment.

Below is the pin-out for the Mictor connector on the Audio Module (top view of male plug):

GND	1	20	GND
DI11	2	21	DI12
DI21	3	22	DI22
DO11	4	23	DO12
DO21	5	24	DO22
GND	6	25	GND
LOUT0	7	26	ROUT1
GND	8	27	GND
LIN0+	9	28	RIN1+
GND	10	29	GND
LOUT2	11	30	ROUT3
GND	12	31	GND
LIN2+	13	32	RIN3+
GND	14	33	GND
LIN0-	15	34	RIN2–
LIN1-	16	35	RIN3–
X	17	36	X
X	18	37	X
X	19	38	X

Figure 2-7 Mictor Connector Pin-out Diagram

Notes

1) X = not connected
2) ± indicates balanced input
3) DIij or DOij indicates digital input or output; i indicates codec 1 or 2, and j indicates digital line 1 or 2.

2.4.1.2 Audio I/O Cable

If an Audio Module is in use, an audio I/O cable will likely be required to bring audio and digital I/O signals from the Audio Module out to external equipment. The audio I/O cable uses a "Mictor" style connector at one end (see section 2.4.1.1 above); the other end terminates in a 37-pin D-sub connector (female). The 37-pin D-sub is suitable for connection to a standard 4-channel "audio breakout cable" (see section 2.4.1.3 below). Audio breakout cables typically provide RCA phono connections, but are also available in BNC format.

Below is the pin-out for the audio I/O cable (front view of female 37-pin D-sub):

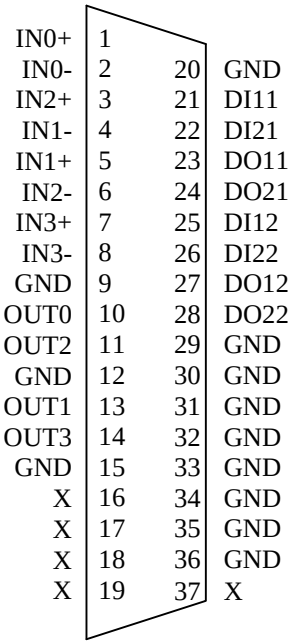


Figure 2-8 Audio I/O Cable Pin-out Diagram

Notes

1) X = not connected
2) ± indicates balanced input
3) DIij or DOij indicates digital input or output; i indicates codec 1 or 2, and j indicates digital line 1 or 2.

2.4.1.3 Audio Breakout Cable

Standard 4-channel "audio breakout cables" are available from Signalogic. 37-pin D-sub male audio breakout cables typically provide RCA phono connections, but are also available in BNC format. Cables have two (2) groups of connectors, labeled A and B, and color-coded in the following format:

Group A	Group B
IN0-white	IN2-white
IN1-red	IN3-red
OUT0-black	OUT2-black
OUT1-yellow	OUT3-yellow

2.4.2 VDS Development System Equipment

Default configuration VDS Development Systems include the following equipment:

Hardware

- SigC54xx Development Board
- SigC54xx-SODIMM module, 100 MHz, 128k x 16 SRAM per processor (can be either 1-, 2-, or 3-processor)

- SigSD4-SODIMM Audio Module, 4-channel 16-bit sigma-delta codec
- Audio I/O Cable
- Optional Audio Breakout Cable (4-channel, either RCA phono or BNC)

Software

- Hypersignal-Macro software
- DirectDSP software
- C54xx Source Code Interface software
- VDS software

3. Software Installation

3.1 DirectDSP Demo Programs

3.1.1 DirectDSP Debugger Demo Program

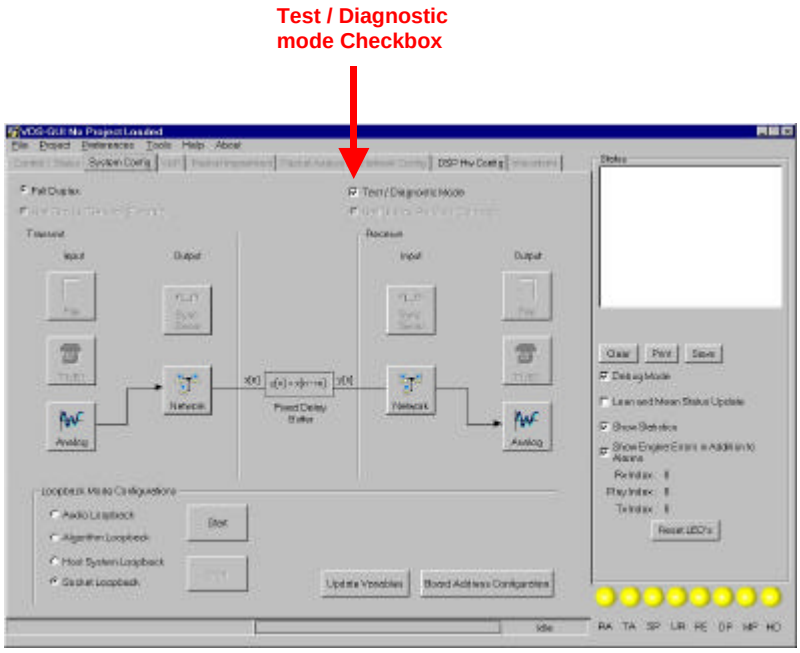
3.2 VDS Software Installation

1) Insert the Signalogic VDS Software CD into your CD-ROM and run "setup.exe" to start the software installation process.
2) Go through the Installation of VDS Software and follow the directions of the host machine (the default values are valid).
3) Installation may require a reboot, if so reboot the host machine. Wait for it to reboot.
4) Once the computer has completely booted (and you have logged onto your system if a network card is installed) click on the Start button, select Programs menu, select Signalogic VDS Software submenu, and click on VDS Software. This will start the VDS program.

4. Software Operation

1) The software will startup on the “System Config” tab. (Note that the only other tab enabled in "Test/Diagnostic" mode is the “DSP HW Config” tab).

2) If the system does not start up on the System Config tab, or does not have the "Test/Diagnostic Mode" checkbox selected, check the “Test / Diagnostic Mode” checkbox.



HELP NOTE 1: Status window will not contain any messages until the test start running.

Figure 4-1 VDS System Configuration Tab

3) If the screen resolution is less than 1280 x 1024 the entire form will not be shown (See HELP NOTE 2 at right for manual screen move instructions)

4) Click on the “DSP HW Config” tab (only other tab enabled in “Test/Diagnostic” mode).

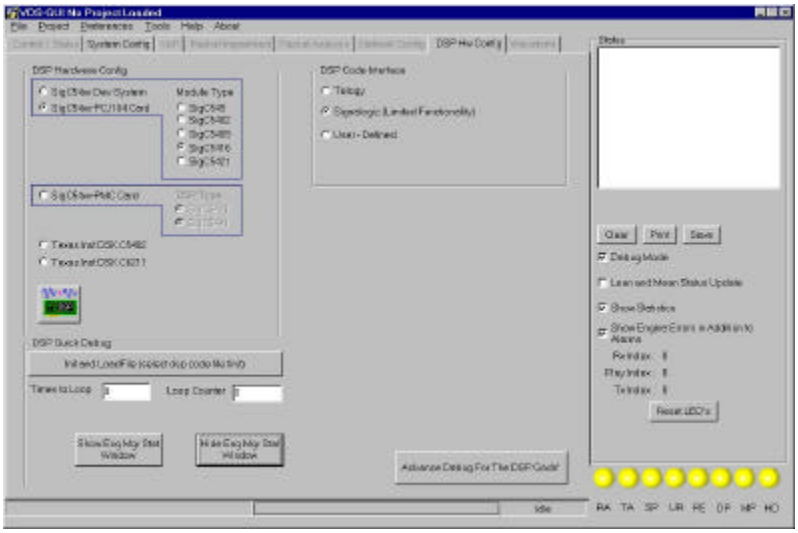


Figure 4-2 The DSP HW Configuration Tab

HELP NOTE 2: (Manual Screen Moving) If the screen resolution is less than 1280x1024, all of the GUI will not show up on the screen. To access the parts of the GUI that are not displayed on the screen (due to low resolution), click the "Preferences" menu, select the "Move Screen" submenu and click "Allow Move", then right click open areas on the GUI and drag to expose hidden the areas. If the GUI needs to be re-centered, the same submenu has a "Re-Center Controls" option. A shortcut to enable the "Move Screen" is SHIFT-F11. To re-center controls has a short cut as well, SHIFT-F12.

5) In the "DSP Hardware Config" frame, click the Hardware Manager Dialog Button (button with the picture of a board and the words "DSP" written on it).

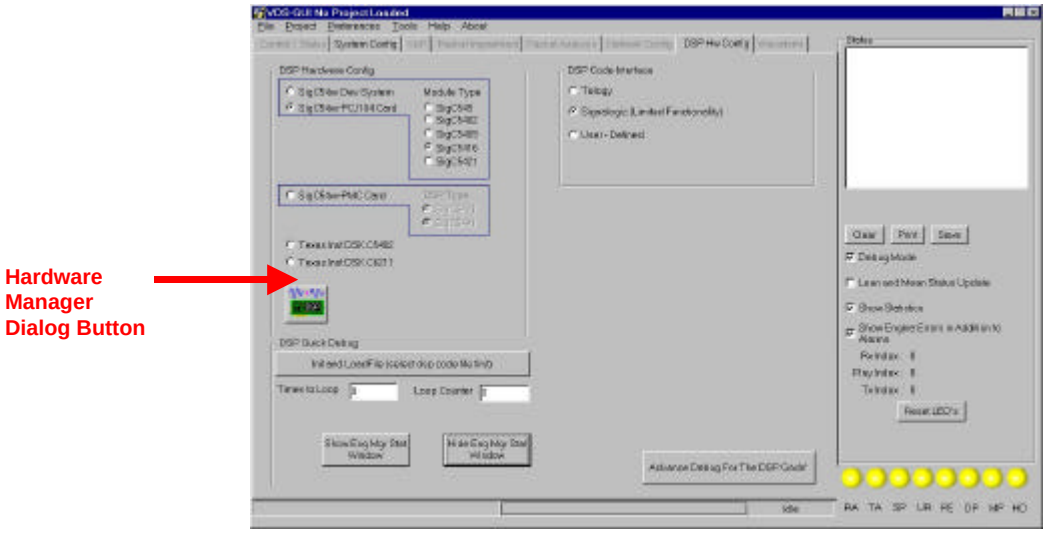


Figure 4-3 DSP Hardware Manager Dialog Button

6) When the Hardware Manager Dialog Box appears, verify that the correct "Setup File" is selected (default filename is HWSetup.lst)

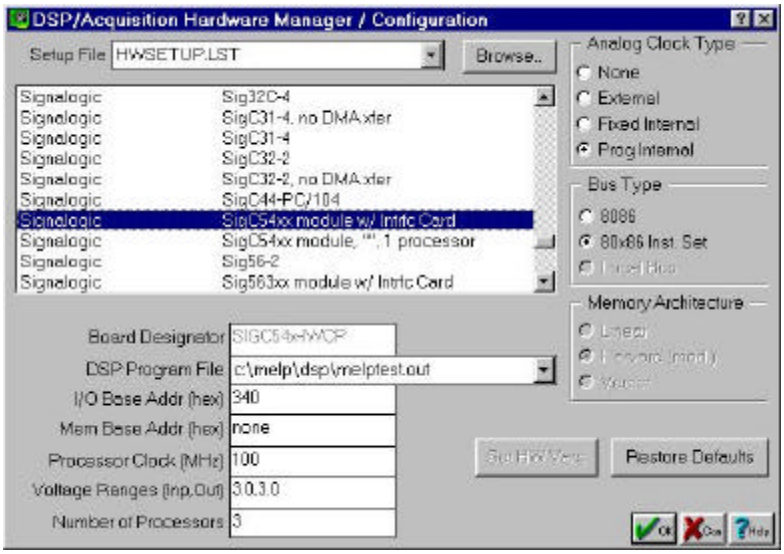


Figure 4-4 DSP Hardware Manager Dialog Box

7) In the List Box, select the correct hardware. (Default for the 3-processor module is "SigC54xx module w/ Interface Card"). If you have a 1-processor module, you should select "SigC54xx Module, "", 1 processor".

8) If you are using a SigC54x/PC104 board, the I/O Base Address field should be ***0x320***, otherwise, it should be ***0x340*** for SigC54xx development system.

9) Verify that the Processor Clock and the Number of Processors fields are set correctly. Default Processor Clock settings are as follows

SigC549	100 MHz
SigC509	100 or 160 MHz
SigC416	100 or 160 MHz

10) Then type in the path to the DSP Program File, if the current one is different (for example, for MELP codec test, the default path is c:\melp\dsp\melptest.out).

11) Click OK to conclude Hardware Configuration and close the Hardware Manager dialog box. This will return to the VDS the "System Config" tab.

12) Right click on the "Analog" button in the Receive frame and select "Properties" from the menu.

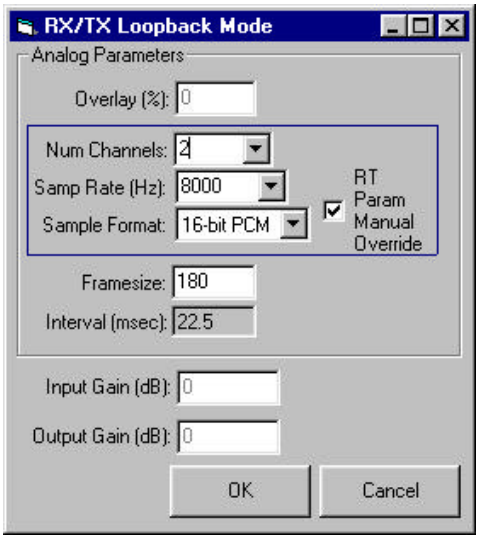


Figure 4-5 Loopback Analog Properties

13) Check all the values and change them if necessary (the defaults shown above in Figure 4-5 should be valid).

14) Click the “Update Variables” button on the “Loopback Mode Configurations” frame.

4.1 Diagnostic Tests in VDS

1) The software will startup on the “System Config” tab. (Note that the only other tab enabled in "Test/Diagnostic" mode is the “DSP HW Config” tab).

2) If the system does not start up on the System Config tab, or does not have the "Test/Diagnostic Mode" checkbox selected, check the “Test / Diagnostic Mode” checkbox.

Test / Diagnostic mode Checkbox

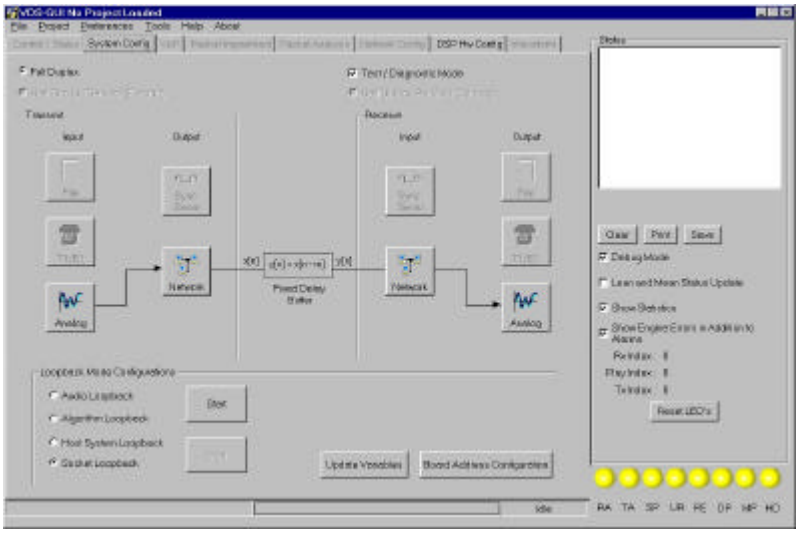


Figure 4-6 VDS System Configuration Tab

3) If the screen resolution is less than 1280 x 1024 the entire form will not be shown (see HELP NOTE 2 in section 4 for manual screen move instructions)

4) Click on the “DSP HW Config” tab (only other tab enabled in “Test/Diagnostic” mode).

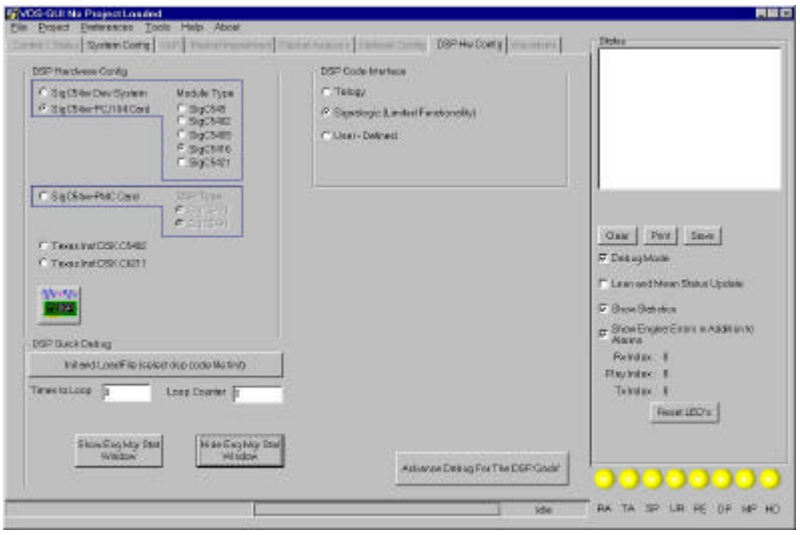


Figure 4-7 The DSP HW Configuration Tab

5) In the “DSP Hardware Config” frame, click the Hardware Manager Dialog Button (button with the picture of a board and the words “DSP” written on it).

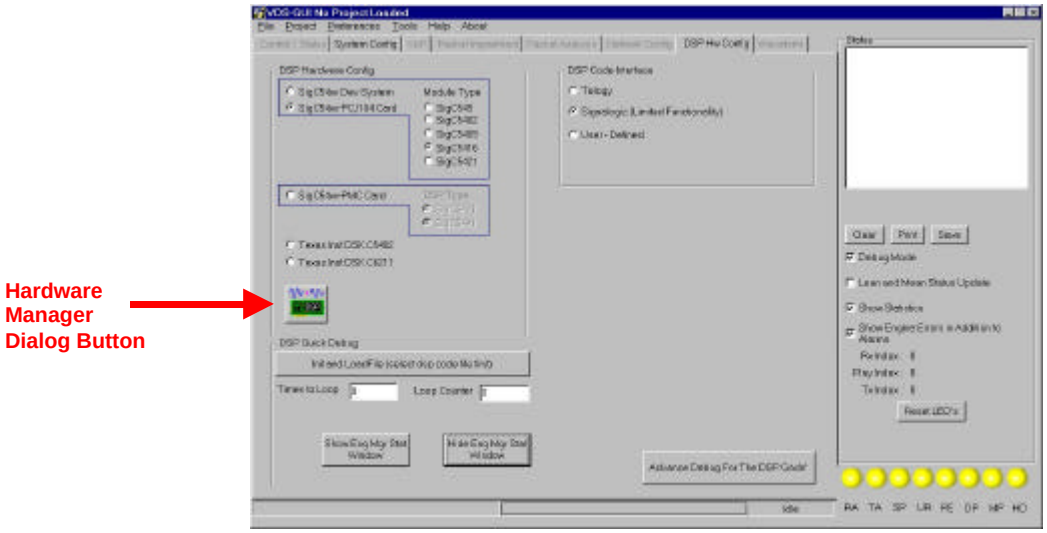


Figure 4-8 DSP Hardware Manager Dialog Button

6) When the Hardware Manager Dialog Box appears, verify that the correct "Setup File" is selected (default filename is HWSetup.lst)

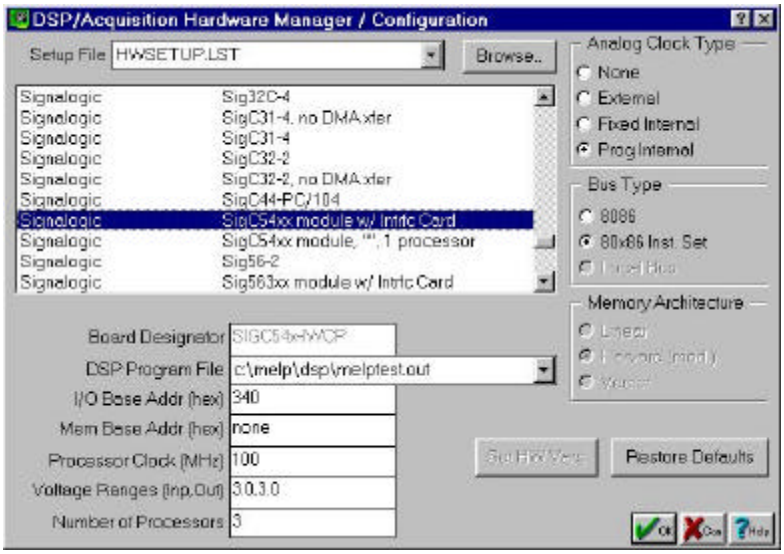


Figure 4-9 DSP Hardware Manager Dialog Box

7) In the List Box, select the correct hardware. (Default for the 3-processor module is "SigC54xx module w/ Interface Card"). If you have a 1-processor module, you should select "SigC54xx Module, "", 1 processor".

8) If you are using a SigC54x/PC104 board, the I/O Base Address field should be ***0x320***, otherwise, it should be ***0x340*** for SigC54xx development system.

9) Verify that the Processor Clock and the Number of Processors fields are set correctly. Default Processor Clock settings are as follows

SigC549 100 MHz

SigC5409	100 or 160 MHz
SigC5416	100 or 160 MHz

10) Then type in the path to the DSP Program File, if the current one is different (for example, for MELP codec test, the default path is c:\melp\dsp\melptest.out).

11) Click OK to conclude Hardware Configuration and close the Hardware Manager dialog box. This will return to the VDS the "System Config" tab.

12) Right click on the "Analog" button in the Receive frame and select "Properties" from the menu.

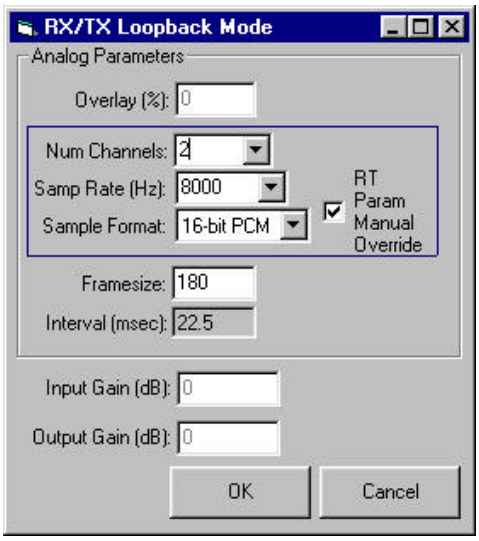


Figure 4-10 Loopback Analog Properties

13) Check all the values and change them if necessary (the defaults shown above in Figure 4-10 should be valid).

14) Click the "Update Variables" button on the "Loopback Mode Configurations" frame.

4.1.1 Low-Level Loopback (Audio Loopback)

1) In this test, the DSP does not perform speech codec processing (either compression or decompression) on audio data. The DSP simply loops data from the audio input to audio output. There is no host action involved in this test other than to read the packet counter incremented by the DSP devices and update in the VDS Status window.

2) Click the "Audio Loopback" radio button in the "Loopback Mode Configurations" frame on the "System Config" tab.

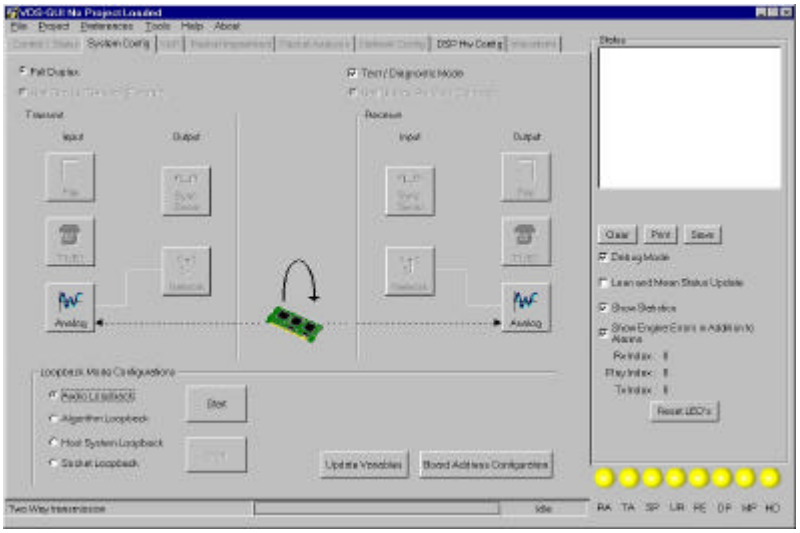


Figure 4-11 VDS GUI with Audio Loopback Selected

3) Click the "Start" button on the "Loopback Mode Configurations" frame and wait for the DSP to initialize and start (this may take up to a minute on fast machines).

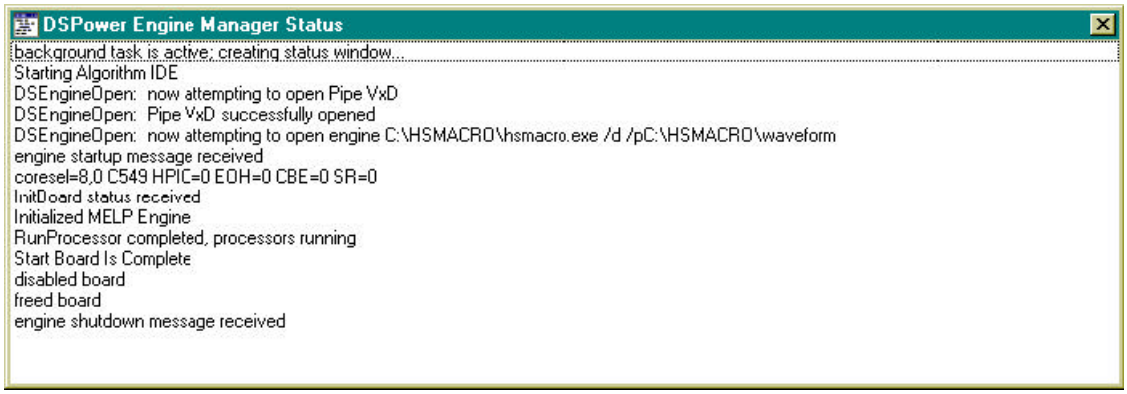

DSPower Engine Manager Status
background task is active; creating status window...
Starting Algorithm IDE
DSEngineOpen: now attempting to open Pipe VxD
DSEngineOpen: Pipe VxD successfully opened
DSEngineOpen: now attempting to open engine C:\HSMACRO\hsmacro.exe /d /pC:\HSMACRO\waveform
engine startup message received
coresel=8,0 C549 HPIC=0 EOH=0 CBE=0 SR=0
InitBoard status received
Initialized MELP Engine
RunProcessor completed, processors running
Start Board Is Complete
disabled board
freed board
engine shutdown message received

Figure 4-12 VDS Engine Manager Status Window

4) You should see the Engine Manager Status window appear with status of the hardware as the hardware initialization process proceeds. (See Figure 4-12 above).

5) **Notes**: Depending upon how many processors are on the installed module(s), you should see corresponding initialization messages in the Engine Manager Status Window. For example, you might see "coresel=8, 0 C549 HPIC=0 EOH=0 CBE=0 SR=0" three (3) times for a 3-processor module.

Engine Manager Status Message	Explanation
coresel=8, 0 C549 HPIC=0 EOH=0 CBE=0 SR=0 coresel=8, 1 C549 HPIC=0 EOH=0 CBE=0 SR=0 coresel=8, 2 C549 HPIC=0 EOH=0 CBE=0 SR=0	Indicates which processors on the DSP module have been initialized

InitBoard status received	Board has been initialized successfully
Initialized MELP Engine	MELP or other algorithm DSP Code file(s) have been downloaded into the DSP(s) successfully
RunProcessor completed, processors running	DSP code is running with appropriate properties set
Startboard Is Complete	At this point, all initialization is done and DSP code is running in real-time. For example for a speech codec; you should be able to hear the audio coming out the speaker assuming you have hooked up the audio input correctly
Disable board	All processors are held in reset and the board is disabled
Freed board	All board handles have been freed
Engine shutdown message received	Indicates the engine has been shut done successfully

Table 4-1 Status Message Explanations

6) Also you should see a packet interval time measurement (for example, approximately 22.5 msec for standard MELP speech codec) and frame counter updated in the VDS Status window and in the Rx Index, Tx Index and Play Index fields below the Status window.

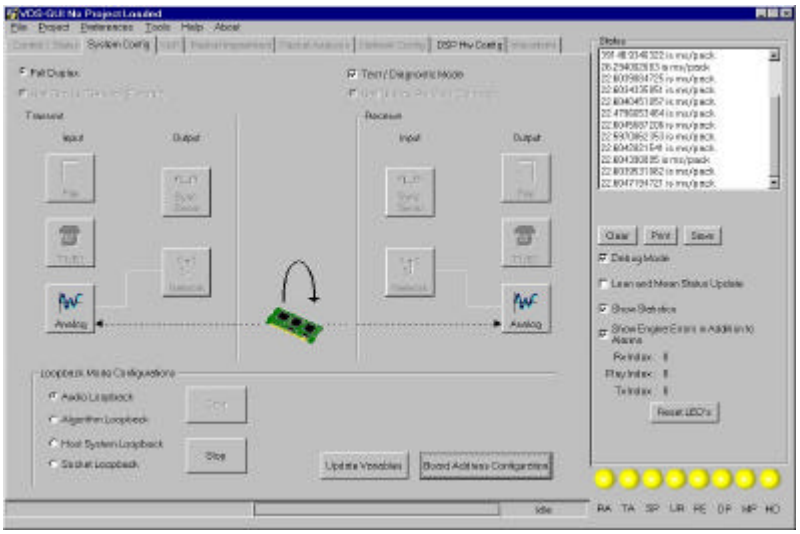


Figure 4-13 VDS Status Window Display with Audio Loopback Enabled

7) If you apply audio input, for example CD player output, speak into the microphone, etc., there should be audio output delayed by approx 2N where N is the speech codec packet size in msec. As one example, if you speak into a microphone you should hear your voice on the speaker output, but with a barely noticeable delay.

8) When you wish to stop, click the “Stop” button on the “Loopback Mode Configuration” frame (this may take up to a minute on fast machines).

4.1.2 Algorithm Loopback

1) In this test, the DSP applies the speech codec or other selected algorithm(s) and processes audio data, looping audio input back to audio output. For example, in the case of a speech codec algorithm, audio input data is compressed and immediately decompressed as audio output data. There is no host action involved in this test other than to read the packet counter incremented by the DSP devices and update in the VDS Status window.

2) Click the “Algorithm Loopback” radio button in the “Loopback Mode Configurations” frame on the VDS “System Config” tab.

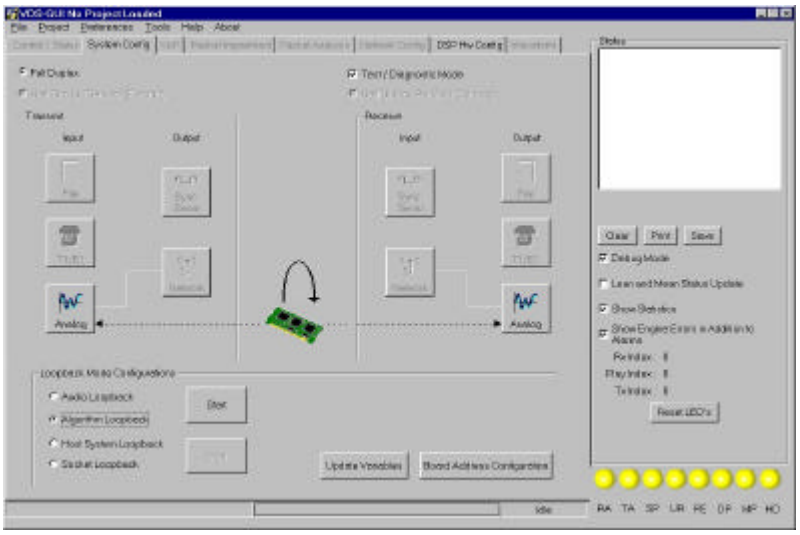


Figure 4-14 VDS GUI with Algorithm Loopback Selected

3) Click the “Start” button on the “Loopback Mode Configurations” frame and wait for the DSP to initialize and start (this may take up to a minute on fast machines).

4) You should see the Engine Manager Status Window appear with the status of the hardware initialization process as it starts up shown there. For a screen capture and detailed information about the window contents, see Figure 4-12 and Table 4-1 above.

5) Also you should see a packet interval time measurement (approximately 22.5 msec for standard MELP speech codec) and frame counter updated in the VDS Status window and in the Rx Index, Tx Index and Play Index fields below the Status window.

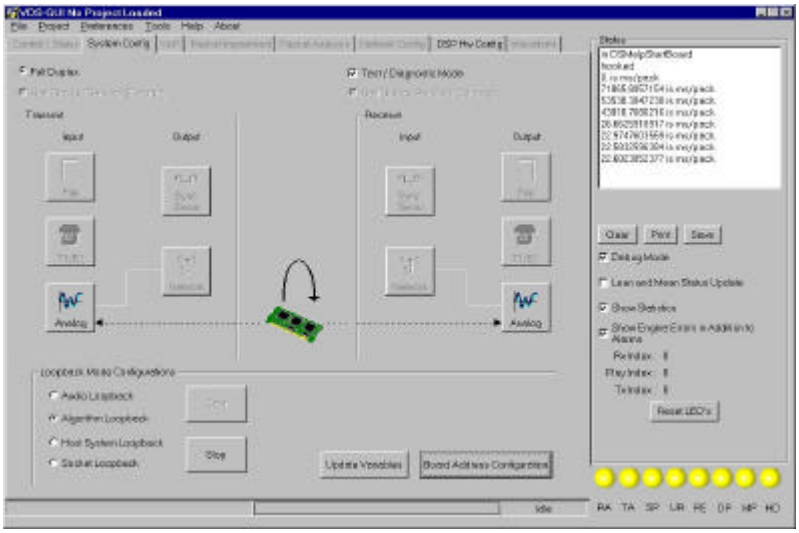


Figure 4-15 VDS Status Window Display with Algorithm Loopback Enabled

6) If you apply audio input, for example CD player output, speak into the microphone, etc., there should be audio output delayed by approx 3N where N is the speech codec packet size in msec. As one example, if you speak into a microphone you should hear your voice on the speaker output, but with a barely noticeable delay.

7) When you wish to stop, click the “Stop” button on the “Loopback Mode Configuration” frame (this may take up to a minute on fast machines).

4.1.3 Host Loopback

1) In this test, the DSP applies the speech codec or other selected algorithm(s) and processes audio data, but performs no loopback. For example, in the case of a speech codec algorithm, audio input data is compressed and stored for subsequent host read, and data written by the host is decompressed as audio output data. Instead of DSP-level algorithm loopback, the host software reads processed audio input packets and loops them back as audio packets. The host software continues to process "packet ready" messages from the DSP devices and real-time operation is fully maintained.

2) Click the “Host System Loopback” radio button in the “Loopback Mode Configurations” frame on the “System Config” tab.

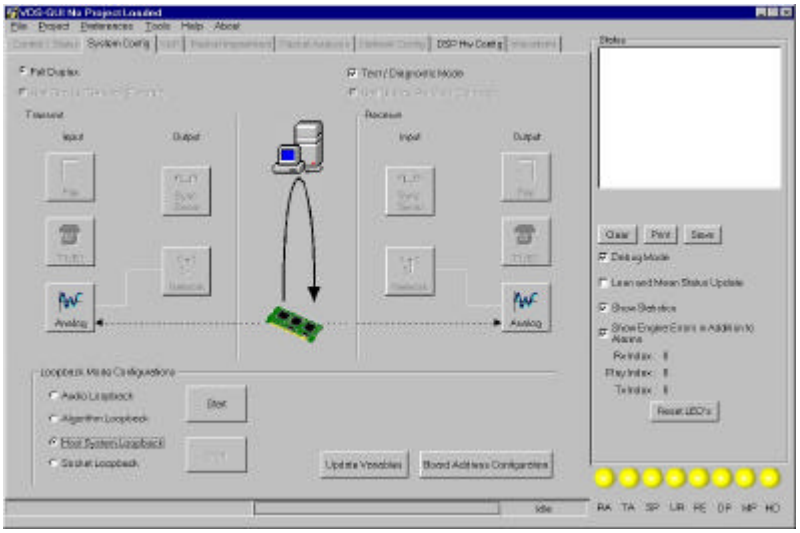


Figure 4-16 VDS GUI with Host System Loopback Enabled

3) Click the “Start” button on the “Loopback Mode Configurations” frame and wait for the DSP to initialize and start (this may take up to a minute on fast machines).

4) You should see the Engine Manager Status Window appear with status of the hardware initialization process as it proceeds. For a screen capture and detailed information about the window contents, see Figure 4-12 and Table 4-1 above.

5) Also you should see a packet interval time measurement (for example, approximately 22.5 msec for standard MELP speech codec) and frame counter updated in the VDS Status window and in the Rx Index, Tx Index and Play Index fields below the Status window.

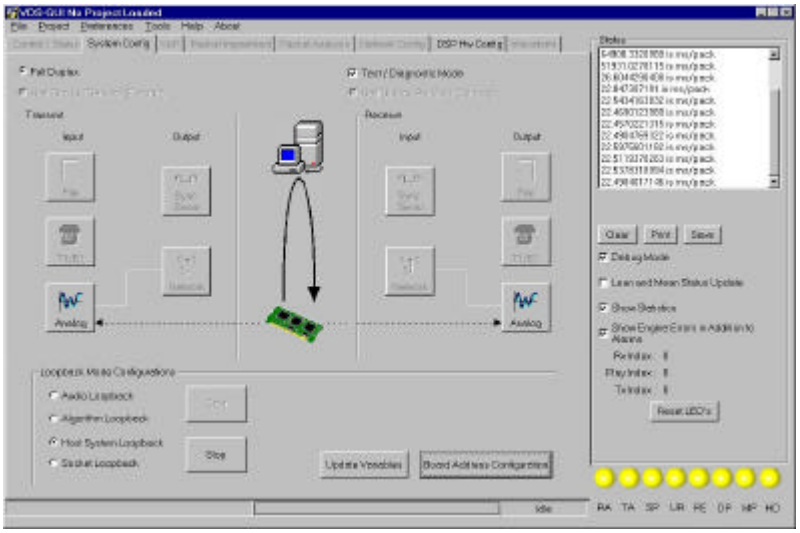


Figure 4-17 VDS Status Window Display with Host System Loopback Enabled

6) If you apply audio input, for example CD player output, speak into the microphone, etc., there should be audio output delayed by approx 3N where N is the speech codec packet size

in msec. As one example, if you speak into a microphone you should hear your voice on the speaker output, but with a barely noticeable delay.

7) When you wish to stop, click the “Stop” button on the “Loopback Mode Configuration” frame (this may take up to a minute of fast machines).

4.1.4 Using More than One Channel

To use more than one channel, simply type in the number of channels you wish to transmit and receive over into the combination box labeled "Num Channels" when setting the Loopback Mode Properties (see step 12 above in sections 4 and 4.1)

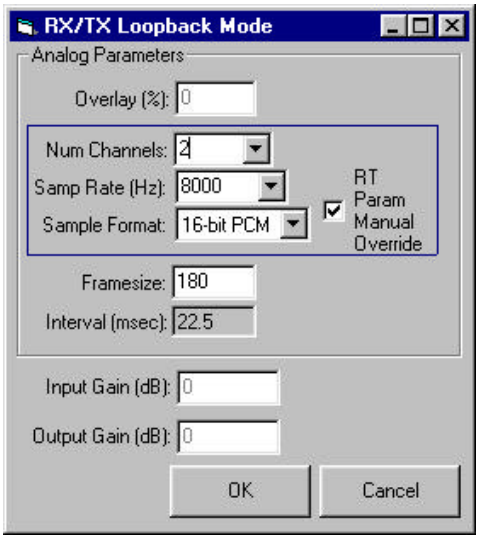


Figure 4-18 Loopback Mode Analog Properties

4.2 Closing VDS

1) To shutdown VDS, click the big ‘x’ on the top right corner of the screen, or click the “File” menu and select “Exit”.

5. Algorithm DSP Software

5.1 DSP Source Code Interface

5.1.1 DSP Coding Example Applications

Some typical examples of applying C54xx Source Code Interface software are listed below. In each case, the application is mentioned, followed by a suggested general approach to make source code modifications.

- Application: performing custom algorithm or feature extraction

  Method: add as application-specific C code (see section 5.1.2 below)

- Application: interfacing to custom or application-specific analog I/O hardware

  Method: modify the entry-point code in the appropriate ISR (interrupt service routine) source code file; modify the analog I/O initialization code (note that this implies the remainder of the ISR can be left "as-is") in t54beg.asm or a add a file with a routine callable from t54beg.asm

- Application: stream real-time results to hard disk as a waveform file for post-analysis, either with or without application-specific C code processing of the real-time data

  Method: use appropriate setup in DSPower or Hypersignal software; see section 5.1.4 below. Hypersignal software or DSPower-Block Diagram software can be used to display, analyze, and measure the stored .tim or .wav file in both time and frequency domain

- Application: enable a "field debug and measurement" mode in a stand-alone DSP product

  Method: implement a flag in the DSP code to determine whether a host interface is present; if so, then debug and data collection functions can be enabled that are normally not

- Application: adding code to support interboard communication

  Method: modify existing ISR; add user-defined C/asm code to user-defined ISR

- Application: adding code to allow digital I/O to application-specific peripheral devices

  Method: modify existing ISR; add user-defined C/asm code to user-defined ISR

- Application: measuring algorithms or parts of algorithms against a particular real-time sampling rate

  Method: add buffer overrun detection to UserProc routine; see section 5.1.3.1

- Application: measuring MIPS requirements in a real-time system

Method: check ISR buffer pointer value at end of calculation to determine remaining percentage of time left in current input frame; add to end of UserProc routine

- Application: evaluating the effects of finite word-length

  Method: add user-defined function to DSP/math function list in either C or asm code

5.1.2 Adding Application-Specific C Code

The following section focuses on adding application-specific C code as one or more C-callable functions which are called on a "buffer basis"; i.e. are called each time a new buffer of analog I/O data must be processed. This approach integrates the required C code into the existing C54xx framework at specific entry-points. For a more general approach, see section 5.2 below, Interface from Application C Code.

The process of adding application-specific C code is designed to isolate the algorithm represented by the C code from background and housekeeping tasks required to maintain host interface and real-time operation. For example, hardware and code initialization and setup, handling sample interrupts and interfacing to analog I/O hardware, communicating with the host, and managing real-time data buffers are performed via pre-defined host and DSP software modules. These modules have well documented properties and methods accessible from host software and, if necessary, from C54xx initialization code. They can be accessed during setup and initialization and on an ongoing basis to allow dynamic control and configuration.

Maintaining reliable, high-speed data transfer host-interface capability in the face of code changes and analog I/O based real-time operation is a critical feature of the SigC54xx Development System; many software functions depend on it. Host-interface capabilities provided by DSPower programs and Hypersignal software include:

- controlling the DSP board
- board interrogation functions
- downloading executable code (COFF files)
- initializing "DSP properties" in the C54xx code
- transferring blocks of real-data between host and DSP board memory (either while code is stopped or running)

For more information on DSPower and Hypersignal capabilities, please see section 5.1.4, C54xx Source Code Access and Control from Host Programs.

5.1.3 Inserting Application-Specific C Code into the Real-Time Stream

Following is general information about how to add user-defined 'C' routines to real-time DSP code running under Hypersignal-Macro or Hypersignal-Acoustic software, or under C/C++, MATLAB, Visual Basic, or LabVIEW programs using the DSPower software API. Actual DSP 'C' code examples are given and differences between DSP chip families are accounted for with *#ifdef..#endif* constructs in the C code.

There are four (4) important ideas underlying the method for adding user-defined 'C' routines to real-time DSP code that is described in this section. These are:

- **hide details of the DSP chip and DSP board**--especially the analog I/O, which tends to vary extensively from board to board and can be extremely complex from a software point of view
- allow user-defined 'C' code to **process analog input/output data on a buffer, or "frame" basis**, and not deal with individual samples (but still allow a single-sample “framesize” in the case of control systems which require minimum system I/O delay
- allow **real-time recording of intermediate results to waveform file** or memory for debugging purposes
- **make the modification cycle, or "turn-around time", as fast as possible**

To achieve these goals, a routine called "UserProc" (in file userproc.c) has been defined in the standard C54xx Source Code which represents a general entry-point for user-defined real-time 'C' code. UserProc is called by the underlying C54xx Source Code framework after each analog input/output data buffer is filled and processed by the analog I/O drivers. This occurs at the interrupt service routine (ISR) level; a flag is set which indicates to foreground (non-interrupt) code that a new buffer is available. The “buffer ready” notification is handled transparently by the C54xx Source Code framework, which forwards the notification from the ISR to foreground C54xx code. Typically, a MODULn .asm file or an application-specific void main() routine running in the foreground receives the notification. In the case of a MODULn function, the notification can be (optionally) forwarded to host-side code, which receives the notification in the form of a callback message or as a release from a blocking WaitForBuffer() function call. Note that the notification occurs *before* analog input buffers are sent to host PC software, such as Hypersignal, or Windows programs using DSPower software (e.g. application-specific Visual C/C++, Visual Basic, MATLAB, or LabVIEW programs). This “insertion point” for application-specific C code allows both analog input buffers (before they reach host PC applications) and output buffers (after they are sent by host PC applications) to be modified by real-time DSP-based C and/or asm code.

5.1.3.1 UserProc: Real-Time 'C' Code Entry Point

Userproc.c represents a general entry-point for user-defined 'C' routines. It is called after each analog input/output data buffer is filled and processed by the analog I/O drivers and has been presented to the foreground processing layer in the C54xx framework. This type of operation is known as "frame-based real-time processing": background interrupt service routines assemble analog input samples into a series of frames, or buffers, which are then made available to

foreground algorithms for processing on a "frame" basis. Note that the "current analog I/O drivers" are drivers that have been selected in the host PC software user-interface for the DSP/analog hardware that is currently installed in the system. In Hypersignal software, this is done using the System Config menu selection; in DSPower programs, this is done using the Hardware Manager dialog box or in DSAssignBoard function calls in the programs.

Parameters passed to userproc.c include a pointer to the just-acquired input buffer, a pointer to the *next* output buffer, length of each buffer in elements, and number of traces (channels) contained in each element. Note that each "element" could actually be more than one sample; for example a three-channel acquisition would cause buffers to contain interleaved data in the form ch0, ch1, ch2, ch0, ch1, ch2, etc.

Below is the default source code listing for the basic userproc.c routine; the default routine is configured for simple loopback operation (loop input back to output) before user-defined modification.

Default UserProc Routine

```
void UserProc(void* ptrIn, void* ptrOut, long nLen, short int nNumTrace) {

#ifdef defined(TMS320C3x) || defined(TMS320C4x) || defined(DSP56xxx) || defined(ADSP2106x)
  #define x ((long*)ptrIn)
  #define y ((long*)ptrOut)
#endif

#ifdef defined(DSP32C) || defined(DSP3210) || defined(TMS320C2x) || defined(TMS320C5x) || defined
(TMS320C54xx) || defined(ADSP21xx)
 #define x ((short int*)ptrIn)
 #define y ((short int*)ptrOut)
#endif

short int  n;

    for (n=0; n<nLen; n++) {

        y[n] = x[n];  /* loop input back to output */

#ifdef defined(TMS320C3x) || defined(TMS320C4x)
        y[n] <<= 16;
#endif
    }
}
```

Here are some important notes about the above code structure:

1) Each input buffer ("x") contains 16-bit A/D values represented as right-justified, signed, N-bit integer values, where N is the natural integer word-length of the DSP device.

2) Each output buffer ("y") contains 16-bit D/A values represented as right-justified, signed, N-bit integer values, where N is the natural integer word-length of the DSP device.

3) The nNumTrace parameter usually corresponds to the number of channels. However, it is more accurate to say "traces" rather than "channels" because of the logical-to-physical channel mapping allowed by the Channel List property specified in host-side software (Hypersignal or DSPower software). The Channel List property allows an arbitrary ordering

of channels; for example, channel 0 may not correspond to trace zero (0).

If the nNumTrace parameter is > 1, then data is stored as interleaved values, in repeating groups of nNumTrace values. For example,

tr0,tr1,tr2,tr0,tr1,tr2, ...

for 3-trace acquisition or signal output.

4) The xxxLINKC.CMD (or xxxLINKC.CTL) linker command (or control) file should be used if run-time 'C' support is needed (see section 5.1.7 below, C54xx Source Code Modification Process). Otherwise, the xxxLINK.CMD or xxxLINK.CTL linker command/control file versions can be used (without C run-time support). See section 5.1.7 below, C54xx Source Code Modification Process, for details. Run-time 'C' support is made necessary by adding functions or routines to the C code which require math and other support libraries. Typically, the library file "rtsPP.lib" would be required, where PP is a name assigned by the DSP chip manufacturer (for example “54” for C54xx, "30" for C3x, "56" for 56xxx, etc.). The xxxLINKC.CMD and xxxLINKC.CTL command/control files link in the appropriate rtsPP.lib file.

5) ***Important Operational Note:*** UserProc must complete its processing *before* the next input/output buffer pair is ready! Otherwise, real-time operation does not occur.

5.1.3.2 UserProc Application Examples

The UserProc entry-point represents a method to integrate application-specific C code into the existing C54xx real-time framework at specific points, or “hooks”. The C code takes the form of “callback” functions which are called by the C54xx framework on a “buffer basis”; i.e. are called each time a new buffer of analog I/O data must be processed. For a more general approach, see section 5.3 below, Interface from Application C Code. A few examples of UserProc functions are given below.

Signal Arithmetic Example

A very simple example of user-defined real-time 'C' code is basic signal arithmetic requiring no recursive input (i.e. no input from previous output samples). In this example, the DSP algorithm scales and adds a constant to input channel 0 data, and also loops the data back to the output. The difference equation for this example could look something like:

```
y[n] = 0.75*x[n] + 1000
```

First, userproc.c is modified to look like:

```
void UserProc(void* ptrIn, void* ptrOut, long nLen, short int nNumTrace) {

#ifdef defined(TMS320C3x) || defined(TMS320C4x) || defined(DSP56xxx) || defined(ADSP2106x)
  #define x ((long*)ptrIn)
  #define y ((long*)ptrOut)
#endif
```

```
#ifdef defined(DSP32C) || defined(DSP3210) || defined(TMS320C2x) || defined(TMS320C5x) || defined
(TMS320C54xx) || defined(ADSP21xx)
 #define x ((short int*)ptrIn)
 #define y ((short int*)ptrOut)
#endif

short int  n;

    for (n=0; n<nLen; n++) {

        x[n] = 0.75*x[n] + 1000;
        y[n] = x[n];

#ifdef defined(TMS320C3x) || defined(TMS320C4x)
        y[n] <<= 16;
#endif
    }
}
```

Second, the following steps are performed:

- Compile userproc.c. As an example of how to do this, the batch file "c.bat" is included which invokes the manufacturer's 'C' compiler from a DOS command prompt.
- Run the **lnk.bat** file if run-time support is *not* needed; i.e. if the rtsPP.lib file is not needed (see section 5.1.7 below, C54xx Source Code Modification Process). Otherwise, run the **lnkc.bat** file.
- Copy the resulting .out file to the Hypersignal directory. The default .out filename is as specified in the Hypersignal-Macro Series Hardware Reference Guide; other filenames can be specified by providing the lnk.bat or lnkc.bat files a filename parameter, for example:

  ```
  lnkc test.out
  ```
  or
  ```
  lnk test
  ```

  which will both produce an output file of "test.out".
- 4) Run host PC software to activate and test newly modified DSP code, using Hypersignal software or DSPower program (i.e. C/C++, MATLAB, Visual Basic or LabVIEW program that calls DSPower functions).
  - **To activate UserProc in Hypersignal operation, enter:**

    ```
    'a,rt,r'
    ```
    or
    ```
    'a,rtd,r'
    ```

    in the ACQ/GEN field in the Analog Conversion function. The 'rt' entry causes real-time simultaneous input/output operation (mode 6; see section 5.1.5 below, C54xx Source Code Interface Structure). The 'rtd' entry works the same way, except recording to waveform file is enabled. Maximum sampling rates at which continuous operation (no gaps or data discontinuity) can be achieved are lower when file recording is enabled. In all cases, maximum sampling rates at which real-time operation can be maintained are heavily system dependent. Important

factors include the DSP chip speed, DSP onboard SRAM access-time and efficiency of user-defined 'C' code. In the case of waveform file recording, factors also include host PC speed, disk drive speed and controller type, ISA or PCI bus speed, etc. For a detailed discussion of the factors influencing maximum continuous sampling rate to file, please see the "Super Guide" online help file which ships with DSPower software products.

Important Note: if the name of the DSP program file has been changed from the default (by renaming the output of lnkc.bat, or by giving lnkc.bat a filename parameter), Hypersignal must know about the change before running Analog Conversion. To do this, suffix the filename to the ANALOG CONVERSION field in the System Config menu. For example, this could result in entries like 'IIC31-B-50,test.out', 'TBS56-B,AC3.OUT', or 'sig32c,dsp32cxx.out'.

- **To activate UserProc when using DSPower software, there are two methods:**
  - **DSAcquireWvfrmFile Approach.** Use either "rt" or "rtd" as the stimulus filename szStimName structure member) in the CONVERSIONINFO structure passed to the DSAcquireWvfrmFile function. This instructs Hypersignal to replace stimulus filename output with real-time processed data. The "rt" value causes indefinite real-time operation (until the engine is idled; see the dtape.cpp source code file for an example of how to use the DSSendEngineCommand(hEngine, DS_SEC_IDLE, DS_EEF_SYNC) function call). The "rtd" value causes real-time operation with continuous recording to waveform file of the analog input data until the NumSamples structure member is reached. The record-to-disk option is provided as a means to post-analyze, in detail, data that has been processed in real-time by userproc.c. Waveform file data can be analyzed using a Hypersignal or DSPower-Block Diagram waveform display function.

    Also, the stimulus mode (szStimMode stucture member) value should be set to "R", which instructs the DSP code to perform repetitive frame output until the waveform file is complete or the operation is aborted.
  - **DSP Code Control Approach.** Modify the rtcode.cpp file as needed. The rtcode.cpp file shows an example of initializing DSP code for UserProc operation with DSSetDSPProperty calls. The key steps are to choose operating mode 6 (set DSP_OPMODE property to 6) and initialize the value of the DSP_FRMSIZ property to the length of each analog input buffer. Normally the DSP_FRMSIZ value should be the same as the DSP_BUFLEN property. Note that with this approach, continuous recording of input or output buffer data to waveform file is not active; to make this functionality active, it would have to be added to the rtcode.cpp file.

Running Sum Example

A slightly more complex example is to combine basic signal arithmetic with recursive input (i.e. input from previous output samples). In this example, the DSP algorithm performs a running

sum on input channel 0 data, with no loopback to the output. The difference equation for this example could look something like:

```
    y[n] = y[n-1] + (x[n] - x[n-M])/M
```

where M is the length of the running sum, in samples. First, userproc.c is modified to look like:

```
#define N 4096        /* define N as maximum buffer size */
#define M 32          /* define M as length of running sum */

short int ySum[N];    /* create storage of size of one buffer, assume it
                         is initialized to zero */

short int xPrev[M];   /* create storage of length of running sum, to hold
                         samples at end of each previous buffer ("memory"
                         between successive frames) */

void UserProc(void* ptrIn, void* ptrOut, long nLen, short int nNumTrace) {

#ifdef defined(TMS320C3x) || defined(TMS320C4x) || defined(DSP56xxx) || defined(ADSP2106x)
  #define x ((long*)ptrIn)
  #define y ((long*)ptrOut)
#endif

#ifdef defined(DSP32C) || defined(DSP3210) || defined(TMS320C2x) || defined(TMS320C5x) || defined
(TMS320C54xx) || defined(ADSP21xx)
 #define x ((short int*)ptrIn)
 #define y ((short int*)ptrOut)
#endif

short int     n,xn;
long static   lSum = 0;

    for (n=0; n<nLen; n++) {

        if (n - M < 0) xn = xPrev[n];    /* use "memory" if index is before
                                            start of current buffer */
        else xn = x[n-M];

        lSum += x[n] - xn;               /* calculate x term sum */

        ySum[n] += (short int)(lSum/M);  /* calculate running sum */
    }

    for (n=0; n<M; n++) xPrev[n] = x[nLen-n];  /* store last M points
                                                  for use by next frame */

    for (n=0; n<nLen; n++) x[n] = ySum[n];     /* store processed data
                                                  in input buffer */
}
```

After modifying userproc.c, see the steps outlined in section 5.1.7 below, C54xx Source Code Modification Process.

Notes

1) The #define constructs above for non-C54xx DSP types can be removed.

2) The above use of a long division intermediate calculation above (lSum/M) is not suggested for real-time processing; it is used only to make the example general and portable across

different DSP devices. A better approach would be to make M a power of 2, so the division could be achieved by shifting the lSum term right.

5.1.4 DSP Source Code Access from Host Programs

There are three (3) general methods of accessing C54xx source code from host programs:

1) Hypersignal-Macro (and Hypersignal-Macro Ex and Hypersignal-Acoustic) software
2) DSPower software
3) DSPower-Block Diagram and Real-Time Composer software

The first two (2) methods above allow flexible user-specified access and control of the DSP(s) at all levels, including debug and measurement functions; these are covered in the sections below. The third method automates DSP hardware control based on graphical signal flow diagrams; the Real-Time Composer software is currently in beta release, and so is not covered in this document at this time.

5.1.4.1 DSP Source Code Access and Control from Hypersignal Software

The Hypersignal-Macro series software packages contain capabilities that facilitate access and control of Hypersignal DSP source code containing user-defined modifications or additions, including:

1) The ability to specify different executable DSP files to load and run when a DSP-based function or display is invoked from Hypersignal. In the System Configuration menu, entering a filename, such as "new.out", after the board designator will override the default DSP filename that Hypersignal uses for default operation.
2) Macro language functions including InitBoard, LoadBoard, RunBoard, ResetBoard, WriteDSPVal and WriteDSPMem that allow macro programs to access DSP/data acquisition hardware as needed.
3) Instrumentation functions which allow record-to-file and display of real-time data from the DSP/data acquisition hardware. For example, a mode can be selected in the Analog Conversion function that allows buffers, either raw or processed, to be uploaded from the shared memory on the DSP board and stored continuously to disk file. For detailed information on this procedure, see section 5.1.3.2, UserProc Application Examples.

In general, maintaining the host interface capability in the C54xx source code allows Hypersignal functions to continue to be enabled and usable. This allows Hypersignal to store data in .tim and .wav waveform file formats, and display and analyze stored data. Time domain and frequency domain displays, DSP operations such as FFT, filtering, difference equations, etc. continue to be accessible with results produced by the C54xx code. See section 5.1.5.2 above, C54xx Source Code Interface Host PC Communication, for more information.

For more information about the functions listed in item 2) above, see Hypersignal-Macro Series Users Manual section 7.10, Macro Commands and Functions.

Notes

1) Both the WriteDSPMem and WriteDSPVal functions always make full-word accesses, depending on the natural word-length of the DSP device. This can affect the address value used, if the DSP is capable of accessing data pieces smaller than full-word length.

2) Both WriteDSPMem and WriteDSPVal functions are capable of accessing all memory types supported by the DSP chip. For a complete list of internal Hypersignal values that can be written to arbitrary DSP source code memory, see Signalogic DSP Software Reference Guide section 3.3, Hypersignal DSP Source Code Variable List. For fully detailed information about the syntax and usage of these functions, see Hypersignal-Macro Series Users Manual section 7.10, Macro Commands and Functions.

5.1.4.2 DSP Source Code Access and Control from DirectDSP Programs

DSPower software is an interface layer between host application programs and DSP/data acquisition hardware and DSP algorithm code. On the application side, it can be incorporated as DLLs and ActiveX controls for programs written in Visual C/C++ and Borland C/C++, MATLAB, Visual Basic, and LabVIEW. On the DSP side, it communicates with drivers for more than 100+ types of DSP/data acquisition hardware; also, it interfaces to Hypersignal-Macro software, MATLAB, and other user-defined drivers and "DSP engines".

DSPower software has two (2) main purposes:

- provide a direct, flexible, but still uniform interface to DSP/data acquisition hardware from popular host environments
- provide a well documented method to interface to DSP algorithms executing in real-time

It is the latter purpose that is the focus of this section. When communicating with DSP code from one of the supported host environments, DSPower software provides the following low-level DSP code-oriented functions:

- download DSP program files (COFF files) produced by the DSP vendor's compiler, assembler, and linker tools
- set and get DSP properties, both pre-defined and user-defined
- transfer data buffers between host memory and onchip and offchip memory on the DSP board, including various data formats, automatic format conversion options, and the ability to transfer data while DSP code continues to execute in real-time

- use COFF symbol information to determine physical addresses of variables, buffers, and vectors in the DSP code
- provide debug capability, including breakpoint, single-step, processor status display, and data display in text or graphical waveform formats

In addition, DSPower software provides the following high-level DSP code oriented functions:

- real-time buffer management, including “buffer ready” messages and callback message processing
- pre-defined instrumentation functions such as digital scope, spectrum analyzer, streaming to disk
- full-featured analog I/O support when applicable

Note that the debug capability mentioned above is not hardware based as with standard JTAG debuggers. Rather, this capability depends on software “trap” interrupts to handle breakpoints and single-step, so it is not “true software-independent debug”. However, unlike JTAG, the DSPower debug capability can transfer large amounts of data without stopping the processor(s).

There are two main approaches to running DSP code from DSPower programs:

- “talker only”
- use pre-defined default Signalogic DSP framework as all or some part of the algorithm code

The “talker only” approach refers to the most general case: only a small interrupt driven host communication kernel (a talker; see section 5.1.5.2, C54xx Source Code Interface Host PC Communication) is used and all other DSP code is application-specific. In this case, no assumptions can be made by DSPower programs about pre-defined properties, buffer locations, code initialization, analog I/O drivers and pre-defined operating modes. This allows DSPower core functionality (COFF download, data transfer, board control and interrogation, etc.) to remain intact, but disables higher level DSPower functions such as buffer management, instrumentation functions such as digital scope, spectrum analyzer, streaming to disk, etc.

Note that the latter case, where the Signalogic C54xx framework is utilized, requires the talker to be enabled, and depends on it for host communication.

The sections below provide a summary of the DSPower API, and also show code examples for several common DSP board functions, including for COFF file download, DSP property initialization, and several types of data transfer. Source code examples are provided for host environments C/C++, MATLAB and Visual Basic.

5.1.4.2.1 DirectDSP API Summary

Below is a partial summary of function calls in the DSPower software package:

Low-Level Functions

Board and processor control, board register and peripheral access, block memory transfers, COFF file download, IEEE-to-DSP conversion

DSInitBoard
DSResetBoard, DSResetProcessor
DSRunBoard, DSRunProcessor
DSHoldBoard, DSHoldProcessor
DSDisableBoard

DSLoadFileProcessor, DSLoadFileBoard

DSPutMem
DSGetMem
DSSetDSPProperty
DSGetDSPProperty

DSWriteBoardReg
DSReadBoardReg

DSIEEEToDSP
DSDSPToIEEE

Autocalculation and Autodetection

autodetect board memory size and architecture, calculate neares allowable sampling frequency

DSGetBoardClass
DSCalcSampFreq
DSGetMemSize
DSGetMemArch
DSGetWordLength

Hardware Management

hardware driver list management, pop-up hardware manager dialog box

DSShowHardwareSelectorDialog

DSGetHWMgrNumEntries
DSGetHWMgrEntryIndex

DSGetHwMgrEntry, DSSetHWMgrEntry

High-Level Functions

record .TIM waveform files, playback .TIM and .WAV waveform files, activate real-time filtering, stimulus & response measurement, continuous signal generation, exec arbitrary DSP functions

DSAcquireWvfrmFile(CONVERSIONINFO FAR*)
DSGenerateWvfrmFile(CONVERSIONINFO FAR*)

CONVERSIONINFO Structure

Buffer and Semaphore Management

buffer-ready message callback window registration, wait for buffer

DSRegisterEngineMsgWnd
DSWaitForBuffer
DSWaitForFlag
DSCancelWait

DSGetBufferInfo
DSGetFlagInfo

Callback Messages

buffer- and flag-ready, status/error, and parameter callback messages

WM_DSPENGINE_BUFRDY
WM_DSPENGINE_FLAGRDY

WM_DSPENGINE_ENGINESTATE
WM_DSPENGINE_FUNCTIONERR
WM_DSPENGINE_CODEGENERROR
WM_DSPENGINE_COMPILEERROR

WM_DSPENGINE_RUNTIMEERROR

WM_DSPENGINE_BUFSIZE
WM_DSPENGINE_SAMPFREQ
WM_DSPENGINE_BUFNUM
WM_DSPENGINE_AMPLEVEL

Engine Management

Driver and engine management, error/status window & reporting

DSEngineOpen
DSEngineClose
DSGetEngineErrorStatus
DSGetEngineUsage
DSEngineSendCommand
DSSetEngineWaveformPath
DSSetEngineTemplatePath

DSShowEngMgrStatusWindow
DSAddEngMgrStatusLine
DSHideEngMgrStatusWindow

Board and Processor Management

Board and processor handle allocation, get board info, get/set board I/O and mem. base addresses, get/set bus type

DSAssignBoard
DSFreeBoard

DSGetBoardInfo

DSAssignProcessor

```
waveform filename
number of channels, channel list
sampling rate
number of samples
output start sample index
gain / attenuation list
digital offset and digital scale
output repeat
input loopback
real-time filter 1 filename
real-time filter 2 filename
trigger mode, level, and delay
number of trigger channels,
trigger channel list
stimulus filename, mode, and delay
```

```
DSFreeProcessor

DSGetBoardBaseAddr, DSSetBoardBaseAddr
DSGetBoardBusType, DSSetBoardBusType

Memory Function Constants

DSGetMemArch,DSPutMem, DSGetMem
constants
DS_GMA_LINEAR
DS_GMA_HARVARD
DS_GMA_VECTOR

DS_GM_VECTOR_DATA_X
DS_GM_VECTOR_DATA_Y
DS_GM_LINEAR_PROGRAM
DS_GM_LINEAR_DATA

DS_GM_SIZE8
DS_GM_SIZE16
DS_GM_SIZE24
DS_GM_SIZE32
```

5.1.4.2.2 C/C++ Examples in DirectDSP

The C/C++ code excerpts below show examples of:

- ✓ Hardware Manager dialog box display
- ✓ Driver and board initialization
- ✓ DSP program file download
- ✓ DSP property initialization
- ✓ Waveform generation buffer calculation and download
- ✓ Run board; run one or more processors
- ✓ Buffer-ready callback message and I/O data buffer processing
- ✓ Cleanup (board shutdown)

For complete C/C++ source code examples, see the following examples included with DSPower software:

Source Code Example Program Type	Filename
Debugger example	sdebug.cpp
Digital oscilloscope	dscope.cpp
Digital tape recorder	dtape.cpp
Real-time C code scope	rtcode.cpp
Simultaneous real-time analog I/O	sim_io.cpp
Spectrum analyzer	spcana.cpp
Strip-chart recorder	strip.cpp

Hardware Manager Dialog Box

```
// show hardware manager dialog (Hardware Manager); returns board
// designator string

// note:  if hardware choice is fixed, then skip this call and use
// correct board designator string in DSAssignBoard call below

   if (DSShowHardwareSelectorDlg(NULL, szBoard) == IDCANCEL)
      goto cleanup;
```

Driver and Board Initialization

The driver and board initialization functions below do not include error checking. For complete examples with error checking, please consult dscope.cpp, sim_io.cpp, and other C/C++ example source code files included with DSPower software.

```
// open engine or driver

   hEngine = DSEngineOpen(DS_EO_HSM, NULL, NULL);  // first try to open
                                                   // Hypersignal-Macro or
                                                   // Hypersignal-Macro EX as
                                                   // engine

   if (!hEngine) {

       hEngine = DSEngineOpen(DS_EO_HSA, NULL, NULL);  // if that doesn't
                                                       // work, try
                                                       // Hypersignal-Acoustic
    }

// note:  using a string name instead of the pre-defined “DS_EO_xxx” constants
// above will access a user-defined driver.  Drivers must export certain
// baseline functions required by DSPower software.  Please ask Signalogic
// for the “OEM DSPower Software Development Guide” document.

// assign a board handle:  engine handle, board designator, bus type, IO base
// addr, Mem base addr

   hBoard = DSAssignBoard(hEngine, szBoard, NULL, NULL, NULL);


// initialize the board; make sure it's installed, reset all processors

   fBoardInitialized = DSInitBoard(hBoard);
```

DSP Program File Download

```
// load executable DSP code file (typically a COFF file produced by DSP
// manufacturer's linker)

   DSLoadFileProcessor(hBoard, NULL, 0x01);  // load default file for the
                                             // board type (processor 0 only)

// interrogate engine for board type values

   wBoardClass = DSGetBoardClass(hBoard);


// determine sampling rate ctrl. reg. value, and actual rate (closest rate
```

```
// possible to desired); CalcSampFreq returns ctrl. reg. value directly, uses
// ptr to return actual sampling frequency (in Hz); demo assumes 1 channel

   dwFsMode = DSCalcSampFreq(hBoard, FsDesired, 1, &ChanList, &FsActual);

// get memory architecture

   uMemArch = DSGetMemArch(hBoard);


   if (uMemArch == DS_GMA_VECTOR)
      BufMul = 1;
   else
      BufMul = 2;

// get address of input time domain data

   dwBufferBaseAddr = DSGetDSPPropertyMem(hBoard, DSP_TIMDATAADDR);

// get address of output time domain data

dwOutBufferBaseAddr = dwBufferBaseAddr + BufMul*wBuflen;

// get the memory size, (note that this currently has to be done after LoadFile)

   dwMemSize = DSGetMemSize(hBoard, 0x01);  // processor 0 only


// reset the DSP board (should already be in reset state; processor 0 only)

   DSResetProcessor(hBoard, 0x01);
```

DSP Property Initialization

The DSP property initialization examples below may not be complete for all pre-defined operating modes in the DSP Source Code Interface. Please also consult the Visual Basic and MATLAB examples for possible additional property examples.

```
// initialize DSP properties required for real-time, continuous simultaneous
// analog I/O operation

   DSSetDSPProperty(hBoard, DSP_BOARDCLASS, wBoardClass & 0x0ff);

   DSSetDSPProperty(hBoard, DSP_BOARDSUBCLASS, wBoardClass >> 8);

   DSSetDSPProperty(hBoard, DSP_OPMODE, 6);     // simultaneous I/O is mode 6

   DSSetDSPProperty(hBoard, DSP_FILTTYPE1, 0);  // trace 1 real-time filter off

   DSSetDSPProperty(hBoard, DSP_FILTTYPE2, 0);  // trace 2 real-time filter off

   DSSetDSPProperty(hBoard, DSP_TRIGLEVEL, 0);       // free-run triggering
   DSSetDSPProperty(hBoard, DSP_TRIGCHANLIST, 0);

   DSSetDSPProperty(hBoard, DSP_BUFLEN, wBuflen);    // input buffer size
   DSSetDSPProperty(hBoard, DSP_STMBUFLEN, wBuflen); // output buffer size

    DSSetDSPProperty(hBoard, DSP_STMDATAADDR,
                     dwOutBufferBaseAddr);      // output buffer base address

   DSSetDSPProperty(hBoard, DSP_HOSTBUFNUM, 0); // host buffer number
   DSSetDSPProperty(hBoard, DSP_BUFNUM, 0);     // DSP buffer number

   DSSetDSPProperty(hBoard, DSP_CHANLIST, 0);   // channel list( each channel is
                                                // 4bits, start from left; first
                                                // channel is 0)
```

```
   DSSetDSPProperty(hBoard, DSP_NUMCHAN, 1);    // number of channels

   DSSetDSPProperty(hBoard, DSP_GAINLIST, 0);   // gain list

   DSSetDSPProperty(hBoard, DSP_FSMODE, dwFsMode);  // sampling rate control
                                                    // register (mode value)

   DSSetDSPProperty(hBoard, DSP_FSVALUE, (long)FsActual));  // actual sampling
                                                            // rate (in Hz)


// set output digital scale (multiply) factor and offset(add) to DSP code

   if ( DSGetBoardInfo(hBoard, DS_GBI_DSPWORDLENGTH) == 24) {
      nShift = 7;
      nShift2 = 8;

   }
   else {
      nShift = 0;
      nShift2 = 0;
   }


// set analog input/output scale factors (scale factors are implemented
// in Q-8 in DSP code)

   DSSetDSPProperty(hBoard, DSP_SCALEIN, 256*InputScaleFactor*(1 << nShift));
   DSSetDSPProperty(hBoard, DSP_SCALEOUT, 256*OutputScaleFactor*(1 << nShift));

// set analog input/output offsets

   DSSetDSPProperty(hBoard, DSP_OFFSETIN, InputOffset*(1 << nShift2));
   DSSetDSPProperty(hBoard, DSP_OFFSETOUT, OutputOffset*(1 << nShift2));
```

Waveform Generation Buffer Calculation and Download

```
// calculate buffer size as approximately holding integral number
// of periods of output sine wave, based on actual sampling frequency,
// to form continuous boundary between output buffers (see output waveform
// calculation and download/storage to board below)

// note the value of buffer size also depends on the type of board;
// different boards have different maximum buffer size depending on
// the amount of onchip/offchip memory available; the value we are
// using is typically small in order to allow this demo program to
// work on a wide range of boards; for better streaming, continuous
// data transfer performance, the buffer size should be increased,
// if the board will allow it; for example, the calculation below
// could be made = / 1, to increase both accuracy and performance

   wBuflen = (int)(FsActual / 10);

   buffer1 = (short*)GlobalLock(hBuf1);
   buffer2 = (short*)GlobalLock(hBuf2);

// Create data buffer; note that data must be scaled to be reasonable values
// for D/A converter(s)

   for (n=0; n < wBuflen; n++){
      buffer1[n] = (short)(10000*sin(2*pi*n*1000/FsActual));
      buffer2[n] = (short)(10000*sin(2*pi*n*1000/FsActual));
   }

// pre_load DSP board memory with data buffers;
// both buffers must perloaded for continuous operation.
// note that download depends on whether board has linear or Harvard memory
// architecture
```

```
   if (uMemArch == DS_GMA_VECTOR){ // board has vector memory architecture

      uStatus = DSPutMem(hBoard, DS_GM_VECTOR_DATA_X, dwOutBufferBaseAddr,
                         DS_GM_SIZE16, buffer1, wBuflen);
      uStatus = DSPutMem(hBoard, DS_GM_VECTOR_DATA_Y, dwOutBufferBaseAddr,
                         DS_GM_SIZE16, buffer2, wBuflen);
   }
   else{  // board has linear memory architecture

      uStatus = DSPutMem(hBoard, DS_GM_LINEAR_DATA_RT, dwOutBufferBaseAddr,
                         DS_GM_SIZE16, buffer1, wBuflen);
      uStatus = DSPutMem(hBoard, DS_GM_LINEAR_DATA_RT,
                         dwOutBufferBaseAddr+wBuflen, DS_GM_SIZE16,
                         buffer2, wBuflen);
   }

   GlobalUnlock(hBuf1);
   GlobalUnlock(hBuf2);

   GlobalFree(hBuf1);
   GlobalFree(hBuf2);
```

Run Board; Run One or More Processors

```
// run board/processor examples

   if (fMultiProcessor)
      DSRunProcessor(hBoard, dwProcList);
   else if (fMultiProcessor && fFirstProcActiveOnly)
     DSRunProcessor(hBoard, 0x01);
   else if (fMultiProcessor && fAllProcActive)
     DSRunBoard(hBoard);
   else if (fSingleProcessor)
    DSRunBoard(hBoard);
```

Buffer-Ready Callback Message and I/O Data Buffer Processing

```
case WM_DSPENGINE_BUFRDY:  // message sent by DSP engine or driver when data
                           // buffer is ready

   pBuffer = (short int*)GlobalLock(hBuf);


   if (uMemArch == DS_GMA_VECTOR) {  // vector data memory

      if (nCurBuf == 0)
         uStatus = DSGetMem(hBoard, DS_GM_VECTOR_DATA_X, dwBufferBaseAddr,
                            DS_GM_SIZE16, pBuffer, wBuflen);
      else
         uStatus = DSGetMem(hBoard, DS_GM_VECTOR_DATA_Y, dwBufferBaseAddr,
                            DS_GM_SIZE16, pBuffer, wBuflen);
      }
      else  // linear data/prog memory, or modified harvard arch. with linear
            // data memory
         uStatus = DSGetMem(hBoard, DS_GM_LINEAR_DATA_RT,
                            dwBufferBaseAddr+nCurBuf*wBuflen, DS_GM_SIZE16,
                            pBuffer, wBuflen);

      if (!uStatus)
        DSMessageBox(hwnd,
                     "DSGetMem:  problem with point transfer", szApp, MB_OK);

/* Output the acquired data
   comment the PutMem section if the pre-loaded data (see WinMain above) should
   continue to be output repetitively; otherwise, this section will loop input
   buffers to output, with a one buffer delay
*/
```

```
   if (uMemArch == DS_GMA_VECTOR) {  // vector data memory

      if (nCurBuf == 0)
         uStatus = DSPutMem(hBoard, DS_GM_VECTOR_DATA_X, dwOutBufferBaseAddr,
         DS_GM_SIZE16, pBuffer, wBuflen);

      else
         uStatus = DSPutMem(hBoard, DS_GM_VECTOR_DATA_Y, dwOutBufferBaseAddr,
         DS_GM_SIZE16, pBuffer, wBuflen);

   }
   else { // linear data/prog memory, or modified harvard arch. with linear
            data memory

      uStatus = DSPutMem(hBoard, DS_GM_LINEAR_DATA_RT,
                         dwOutBufferBaseAddr+nCurBuf*wBuflen, DS_GM_SIZE16,
                         pBuffer, wBuflen);

   }

   GlobalUnlock(hBuf);

   if (!uStatus)
     DSMessageBox(hwnd,
                  "DSGetMem:  problem with point transfer", szApp, MB_OK);
   else FirstBufferRcvd = TRUE;

   nCurBuf ^= 1;  // switch buffers

// wait for next buffer; note this function does not actually wait unless the
// DS_WFB_SYNC flag is used; in the example below, it sets up a callback message

   DSWaitForBuffer(hBoard, nCurBuf, NULL, DS_WFB_POLLED);


// update the screen with new data

   MyPlotFunction(hBuf);
```

Cleanup (Board Shutdown)

```
   if (hBoard != NULL) {

      if (fBoardInitialized) DSDisableBoard(hBoard);  // disable board (all
                                                      // processors)
      DSFreeBoard(hBoard);
   }

   if (hEngine != NULL) DSEngineClose(hEngine);
```

5.1.4.2.3 MATLAB Examples in DirectDSP

The MATLAB code excerpts below show examples of:

- ✓ Hardware Manager dialog box display
- ✓ Driver and board initialization
- ✓ DSP program file download
- ✓ DSP property initialization
- ✓ Waveform generation buffer calculation and download
- ✓ Run board; run one or more processors

- ✓ Continuous I/O data buffer processing
- ✓ Cleanup (board shutdown)

For complete MATLAB source code examples, see the following .m file examples included with DSPower software:

Source Code Example Program Type	Filename
Digital oscilloscope	dscope.m
Data format conversion	hwtest3.m
Continuous buffer analog input	hwtest5.m
Continuous buffer analog output	hwtest4.m
Frame-based analog buffer input and plot	hwtest2.m
Simultaneous real-time analog I/O	hwtest6.m
Waveform file playback	hwtest1.m

Hardware Manager Dialog Box

```
% show hardware manager dialog box ("Hardware Manager"); returns board
% designator
% string selected by user
% note:  if hardware choice is fixed, then skip this call and use correct board
% designator
% string in AssignBoard call below

 [fSelect, BoardStr] = ShowHwSelector(NULL);

 if (fSelect == IDCANCEL)  % user canceled?
  return;
 end
```

Driver and Board Initialization

```
% open dsp engine or driver

% default: first try to open Hypersignal-Macro (or Macro Ex) as engine

hEngine = EngineOpen(DS_EO_HSM, NULL, NULL, DS_EO_NOTVISIBLE);

if (hEngine == NULL)

% if that doesn't work, try Hypersignal-Acoustic

    hEngine = EngineOpen(DS_EO_HSA, NULL, NULL, DS_EO_NOTVISIBLE);

    if (hEngine == NULL)

        errstr = ['EngineOpen failed; error code = ' num2str(GetHwLibErrStat)];
        disp(errstr);

        return;  % abort
    end
end
```

```
% note:  using a string name instead of the pre-defined “DS_EO_xxx” constants
% above will open a user-defined driver.  Drivers must export certain
% baseline functions required by DSPower software.  Please ask Signalogic
% for the “OEM DSPower Software Development Guide” document.


% assign a board handle:  engine handle, board designator string, instruction
% set, IO base addr, Mem base addr

hBoard = AssignBoard(hEngine, BoardStr, NULL, NULL, NULL);  % NULL == use
                                                            % default values


% initialize the board; make sure it's installed and responding

fBoardInitialized = InitBoard(hBoard);

if (fBoardInitialized == FALSE)

    errstr = ['InitBoard failed; error code = ' num2str(GetEngErrStat(hEngine))];
    disp(errstr);

    if (hBoard ~= NULL)
        FreeBoard(hBoard);
    end

    if (hEngine ~= NULL)
        EngineClose(hEngine);
    end

    return;  % abort
end
```

DSP Program File Download

```
% load executable DSP code file (usually a COFF file produced by DSP
% manufacturer's linker)

if (LoadProcessor(hBoard, NULL, 1) == NULL)  % load default file for the board
                                             % type (processor 0 only)

    errstr = ['LoadProcessor failed; error code = ' num2str(GetEngErrStat(hEngine))];
    disp(errstr);

    if (hBoard ~= NULL)
        FreeBoard(hBoard);
    end

    if (hEngine ~= NULL)
        EngineClose(hEngine);
    end

    return;  % abort
end
```

DSP Property Initialization

The DSP property initialization examples below may not be complete for all pre-defined operating modes in the DSP Source Code Interface. Please also consult the C/C++ and Visual Basic examples for possible additional property examples.

```
% prepare hardware and data transfer values, allocate memory to receive buffers

BufNum = 0;               % start with buffer 0
NumChan = 1;              % this example uses one channel
InputOffset  = 0.0;       % DSP input offset
```

```
OutputOffset = 0.0;        % DSP output offset
InputScaleFactor  = 1.0;   % DSP input scale factor
OutputScaleFactor = 1.0;   % DSP output scale factor
desiredSampFreq = 20000;   % desired sampling rate
desiredOutFreq = 1000;     % desired output frequency

BufSize = min(desiredSampFreq*5, 4096);  % default buffer size

BoardClass = GetClass(hBoard);       % board classification
MemArch = GetMemArch(hBoard);        % board memory architecture

if (MemArch == DS_GMA_VECTOR)
 BufMul = 1;                         % multiplier for BufSize
else
 BufMul = 2;
end

TimBaseAddr = GetVarMem(hBoard, DSP_TIMDATAADDR);  % base address of input
OutBaseAddr = TimBaseAddr+BufMul*BufSize;          % base address of output

[FsMode, FsActual] = CalcSampFreq(hBoard, desiredSampFreq, NumChan, NULL);

buf = zeros(1, BufSize);       % allocate buffer memory  ***
buf1 = zeros(1, BufSize);
buf2 = zeros(1, BufSize);
buf_plot = zeros(1, BufSize);

ResetBoard(hBoard);  % reset board (should already be in reset state)

% write property values to DSP code on board

SetDSPProperty(hBoard, DSP_BOARDCLASS, rem(BoardClass, 256));
SetDSPProperty(hBoard, DSP_BOARDSUBCLASS, fix(BoardClass/256));

SetDSPProperty(hBoard, DSP_OPMODE, 6);     % continuous input/output mode

SetDSPProperty(hBoard, DSP_FILTTYPE1, 0);  % disable trace 1 real-time filter
SetDSPProperty(hBoard, DSP_FILTTYPE2, 0);  % disable trace 2 real-time filter

SetDSPProperty(hBoard, DSP_BUFLEN, BufSize);    % input buffer size
SetDSPProperty(hBoard, DSP_STMBUFLEN, BufSize); % output buffer size

SetDSPProperty(hBoard, DSP_STMDATAADDR, OutBaseAddr);  % output buffer base
                                                       % address

SetDSPProperty(hBoard, DSP_HOSTBUFNUM, 0);      % host buffer number
SetDSPProperty(hBoard, DSP_BUFNUM, 0);          % DSP buffer number

SetDSPProperty(hBoard, DSP_CHANLIST, 0);        % channel list (each channel is 4 bits, start
from left; first channel is 0)
SetDSPProperty(hBoard, DSP_NUMCHAN, NumChan);   % number of channels

SetDSPProperty(hBoard, DSP_GAINLIST, 0);        % gain/attenuation list

SetDSPProperty(hBoard, DSP_TRIGLEVEL, 0);              % free-run triggering
SetDSPProperty(hBoard, DSP_TRIGCHANLIST, 0);         % triger channel

% set output digital scale (multiply) factor and offset (add) to DSP code

if (GetBoardInfo(hBoard, DS_GBI_DSPWORDLENGTH) == 24)
    nShift = 7;    % weird efficiency thing required by DSP56xxx boards
    nShift2 = 8;
else
    nShift = 0;
    nShift2 = 0;
end

% write analog I/O scale factors (scale factors are implemented as Q-8 in DSP
% code)

SetDSPProperty(hBoard, DSP_SCALEIN,   256*InputScaleFactor*(2^nShift));
```

```
SetDSPProperty(hBoard, DSP_SCALEOUT,  256*OutputScaleFactor*(2^nShift));

% write analog I/O offset values

SetDSPProperty(hBoard, DSP_OFFSETIN,  InputOffset);
SetDSPProperty(hBoard, DSP_OFFSETOUT, OutputOffset*(2^nShift2));

% set sampling rate control register (mode value)

SetDSPProperty(hBoard, DSP_FSMODE, FsMode);

% set actual sampling rate (in Hz)

SetDSPProperty(hBoard, DSP_FSVALUE, FsActual);

% Note:  uncomment the DSP_FRMSIZ property below to cause "userproc.c"
% to be inserted into the real-time data stream processing on the DSP
% board.  This file is included in the "DSP Source Code Interface"
% software, and is intended to serve as an entry point for user-defined
% real-time C (or .asm) code which runs on the DSP board.  This approach
% enables the built-in analog I/O interrupt and buffer processing
% on the board to run normally, initialized from MATLAB, and also keeps
% all host interface and communication routines intact.  This allows
% a focus only on the application-specific, C programming needed for
% real-time operation, and minimizes the amount of programming.  For
% more information, please see www.signalogic.com/ccode.htm.

% SetDSPProperty(hBoard, DSP_FRMSIZ, 1);
```

Waveform Generation Buffer Calculation and Download

```
% create data buffer; note that data must be scaled to be reasonable values
% for D/A converter(s)

for (n=1:BufSize)     % output buffer is 1 Hz sine wave

   buf1(n) = 10000*sin(2*pi*n*1000/FsActual);
   buf2(n) = 10000*sin(2*pi*n*1000/FsActual);

end

% pre-load DSP board memory with data buffers; both buffers must be preloaded
% for continuous operation.  Note that download depends on whether board has
% linear or Harvard memory architecture

if (MemArch == DS_GMA_VECTOR)   % board has vector memory architecture

    Status = PutMem(hBoard, DS_GM_VECTOR_DATA_X, OutBaseAddr, DS_GM_SIZE16_CVT,
                    buf1, BufSize);
    Status = PutMem(hBoard, DS_GM_VECTOR_DATA_Y, OutBaseAddr, DS_GM_SIZE16_CVT,
                    buf2, BufSize);

else  % board has linear or Harvard memory architecture

    Status = PutMem(hBoard, DS_GM_LINEAR_DATA_RT, OutBaseAddr, DS_GM_SIZE16_CVT,
                    buf1, BufSize);
    Status = PutMem(hBoard, DS_GM_LINEAR_DATA_RT, OutBaseAddr+BufSize,
                    DS_GM_SIZE16_CVT, buf2, BufSize);
end

if (Status == NULL)
    errstr = ['problem with PutMem; error code = ' num2str(GetEngErrStat(hEngine))];
    disp(errstr);
end
```

Run Board; Run One or More Processors

```
% run board/processor examples

   if (fMultiProcessor)
      RunProcessor(hBoard, dwProcList);
   else if (fMultiProcessor && fFirstProcActiveOnly) then
     RunProcessor(hBoard, 0x01);
   else if (fMultiProcessor && fAllProcActive)
     RunBoard(hBoard);
   else if (fSingleProcessor)
    RunBoard(hBoard);
```

Continuous I/O Data Buffer Processing

```
% Start board and wait for buffer to be output.  Note above that we filled
% second buffer also, to avoid "screech"--because DSP will start output
% on second buffer before we can stop the processor

% Note:  first buffer will be output as soon as board starts running; for
% continuous % output, "Buffer Ready" callback messages must be processed,
% and the RegMsgWnd() and WaitForBuffer() functions must be used

RunProcessor(hBoard, 1);  % run; processor #1 only

fCont = 10;               % output 10 buffers

while fCont > 0,

% wait to make sure first buffer has been output
  WaitForBuffer(hBoard, BufNum, NULL, DS_WFB_POLLED + DS_WFB_SYNC);

  if (MemArch == DS_GMA_VECTOR)  % board has vector memory architecture

    if (BufNum == 0)
        Status = GetMem(hBoard, DS_GM_VECTOR_DATA_X, TimBaseAddr,
                        DS_GM_SIZE16_CVT, buf_plot, BufSize);
    else
        Status = GetMem(hBoard, DS_GM_VECTOR_DATA_Y, TimBaseAddr,
                        DS_GM_SIZE16_CVT, buf_plot, BufSize);
    end
  else  % board has linear or Harvard memory architecture

    Status = GetMem(hBoard, DS_GM_LINEAR_DATA_RT, TimBaseAddr+BufNum*BufSize,
                    DS_GM_SIZE16_CVT, buf_plot, BufSize);
  end

  buf = buf_plot;  % store acquired data in variable buf for PutMem use

  if (Status == NULL)
    errstr = ['problem with GetMem; error code = ' num2str(GetEngErrStat(hEngine))];
    disp(errstr);
  end

% To change the output during data acquisition, uncomment the following section
% of code, which feeds the input to the output directly for each buffer
%
%  if (MemArch == DS_GMA_VECTOR)  % board has vector memory architecture
%
%    if (BufNum == 0)
%       Status = PutMem(hBoard, DS_GM_VECTOR_DATA_X, OutBaseAddr,
                        DS_GM_SIZE16_CVT, buf, BufSize);
%    else
%       Status = PutMem(hBoard, DS_GM_VECTOR_DATA_Y, OutBaseAddr,
                        DS_GM_SIZE16_CVT, buf, BufSize);
%    end
%    else  % board has linear or Harvard memory architecture
%
%    Status = PutMem(hBoard, DS_GM_LINEAR_DATA_RT, OutBaseAddr+BufNum*BufSize,
                     DS_GM_SIZE16_CVT, buf, BufSize);
%  end
%
%  if (Status == NULL)
%    errstr = ['problem with GetMem; error code = ' num2str(GetEngErrStat(hEngine))];
%    disp(errstr);
%  end

  BufNum = 1 - BufNum;  % toggle buffer number

  fCont = fCont -1;     % to loop forever, do not decrement fCont

end
```

```
% display data

ymax=max(buf_plot);          % set y-axis maximun
ymin=min(buf_plot);          % set y-axis minimum

figure(gcf);                 % use current figure, if one

axis([0, BufSize-1, ymin, ymax]);
pt = min(BufSize, 1000);
plot(buf_plot(1:pt));        % show part of the data
```

Cleanup (Board Shutdown)

```
% done; leave board in disabled state, clean up handles, engines

if (hBoard ~= NULL)

   if (fBoardInitialized == TRUE)
      DisableBoard(hBoard);
   end

   FreeBoard(hBoard);
end

if (hEngine ~= NULL)
   EngineClose(hEngine);
end
```

5.1.4.2.4 Visual Basic Examples in DirectDSP

The Visual Basic code excerpts below show examples of:

- ✓ Hardware Manager dialog box display
- ✓ Driver and board initialization
- ✓ DSP program file download
- ✓ DSP property initialization
- ✓ Run board; run one or more processors
- ✓ Buffer-ready callback message and I/O data buffer processing
- ✓ Waveform data display
- ✓ Cleanup (board shutdown)

For complete Visual Basic source code examples, see the following examples included with DSPower software:

Source Code Example Program Type	**Filename**
Debugger example	sdebug.bas
Digital oscilloscope	dscope.bas
Digital tape recorder	dtape.bas
Spectrum analyzer	spcana32.bas
Strip-chart recorder	strip.bas

Hardware Manager Dialog Box

```
'  show hardware manager dialog (Hardware Manager); returns board designator
' string
' note: if hardware choice is fixed, then skip this call and use correct board
' designator string in DSAssignBoard call below

   If DSShowHardwareSelectorDlg(0, szBoard) = IDCANCEL Then
      GoTo Abort
   End If
```

Driver and Board Initialization

```
' first try to open Hypersignal-Macro or Hypersignal-Macro EX as engine

hEngine = DSEngineOpen(ByVal DS_EO_HSM, ByVal 0&, NUL)

If (hEngine = NUL) Then

' if that doesn't work, try Hypersignal-Acoustic

      hEngine = DSEngineOpen(ByVal DS_EO_HSA, ByVal 0&, NUL)

      If hEngine = 0 Then

         nStatus = DSGetEngMgrErrorStatus()
         If (nStatus = NUL) Then nStatus = DSGetHWLibErrorStatus()
         tmpstr = Str$(nStatus)
         tmpstr = "DSEngineOpen error code =" + tmpstr
         ret = DSMessageBox(NUL, tmpstr, szApp,
                            MB_ICONEXCLAMATION Or MB_ALWAYSONTOP)
         GoTo Abort

      End If

   End If

' note:  using a string name instead of the pre-defined “DS_EO_xxx” constants
' above will open a user-defined driver.  Drivers must export certain
' baseline functions required by DSPower software.  Please ask Signalogic
' for the “OEM DSPower Software Development Guide” document.

' assign a board handle:  engine handle, board designator, bus type, IO base
' addr, Mem base addr

   hBoard = DSAssignBoard(hEngine, ByVal szBoard, 0, 0, 0)

' initialize the board; make sure it's installed, reset all processors

   fBoardInitialized = DSInitBoard(hBoard)

   If (fBoardInitialized = NUL) Then

      nStatus = DSGetHWLibErrorStatus()
      If (nStatus = NUL) Then nStatus = DSGetEngineErrorStatus(hEngine)
      tmpstr = Str$(nStatus)
      tmpstr = "DSEngineOpen error code =" + tmpstr
      ret = DSMessageBox(NUL, tmpstr, szApp,
                         MB_ICONEXCLAMATION Or MB_ALWAYSONTOP)
      GoTo Abort

   End If

' interrogate engine for board type values

   wBoardClass = DSGetBoardClass(hBoard)
```

```
' get memory architecture

   uMemArch = DSGetMemArch(hBoard)

   If (uMemArch = NUL) Then

      ret = DSMessageBox(NUL, "DSGetMemArch failed", szApp,
                         MB_ICONEXCLAMATION Or MB_ALWAYSONTOP)
      GoTo Abort
   End If

' get memory size, (note that this currently has to be done after LoadFile)

   dwMemSize = DSGetMemSize(hBoard, &H1)  'processor 0 only


' reset DSP board (should already be in reset state; processor 0 only)

   ret = DSResetProcessor(hBoard, &H1)


' set the message blaster (MSGBLAST.OCX or MSGBLAST.VBX) to capture buffer ready
' messages from the DSP driver/engine

   Buffer_Ready.hWndTarget = graph_window.hWnd
   Buffer_Ready.AddMessage WM_DSPENGINE_BUFRDY, POSTPROCESS

' register handle of graph window with engine, to receive "buffer ready"
' messages; this is same as registering callback function in C/C++

   ret = DSRegisterEngineMsgWnd(hEngine, DS_REMW_SETDSPDATARDYMSG,
                                graph_window.hWnd)
```

DSP Program File Download

```
' load executable DSP code file (usually a COFF file produced by DSP
' manufacturer's linker)

' load default file for the board type (processor 0 only)

   ret = DSLoadFileProcessor(hBoard, ByVal 0&, &H1)

   If (ret = NUL) Then
      ret = DSMessageBox(NUL, "DSLoadFileBoard: problem loading file", szApp,
                         MB_ICONEXCLAMATION Or MB_ALWAYSONTOP)
      GoTo Abort
   End If
```

DSP Property Initialization

The DSP property initialization examples below may not be complete for all pre-defined operating modes in the DSP Source Code Interface. Please also consult the C/C++ and MATLAB examples for possible additional property examples.

```
' write DSP properties

   ret = DSSetDSPProperty(hBoard, DSP_BOARDCLASS, wBoardClass And &HFF)

   ret = DSSetDSPProperty(hBoard, DSP_BOARDSUBCLASS, Shift(wBoardClass, -8))

   ret = DSSetDSPProperty(hBoard, DSP_OPMODE, 5)  ' spectrum analyzer is mode 5

   If (DSGetBoardInfo(hBoard, DS_GBI_DSPWORDLENGTH) = 24) Then
      nShift = 7    ' weird efficiency thing required by DSP56xxx boards
   Else
```

```
      nShift = 0
   End If

   ret = DSSetDSPProperty(hBoard, DSP_SCALEIN, Shift(256&, nShift))

   ret = DSSetDSPProperty(hBoard, DSP_FILTTYPE1, 0) ' trace 1 real-time filt off

   ret = DSSetDSPProperty(hBoard, DSP_FILTTYPE2, 0) ' trace 2 real-time filt off

   ret = DSSetDSPProperty(hBoard, DSP_TRIGLEVEL, 0) ' free-run triggering

   ret = DSSetDSPProperty(hBoard, DSP_TRIGCHANLIST, 0)

   ret = DSSetDSPProperty(hBoard, DSP_NUMCHAN, NumChan)    ' number of channels

   ret = DSSetDSPProperty(hBoard, DSP_HOSTBUFNUM, 0)  ' starting buffer number

   ret = DSSetDSPProperty(hBoard, DSP_BUFNUM, 0)

   dw = 0
   For i = 0 To (NumChan - 1)
      dw = dw Or Shift(i, 4 * i) ' shift left
      ChanList(i) = i
   Next i

   ret = DSSetDSPProperty(hBoard, DSP_CHANLIST, dw)        ' channel list

   dwFsMode = DSCalcSampFreq(hBoard,
                             FsDesired,
                             NumChan,
                             ChanList(0),
                             dwFsActual)

   ret = DSSetDSPProperty(hBoard, DSP_FSMODE, dwFsMode)  ' sampling rate control
                                                         ' register (mode value)

   ret = DSSetDSPProperty(hBoard, DSP_GAINLIST, 0) ' gain list

   ret = DSSetDSPProperty(hBoard, DSP_FSVALUE, dwFsActual)  ' actual sampling
                                                             ' rate (in Hz)

' Framesize property required for real-time C code data transfer to host
'  operation (mode 6), or Spectrum Analyzer operation (mode 5)

   ret = DSSetDSPProperty(hBoard, DSP_FRMSIZ, Framesize)

   ret = DSSetDSPProperty(hBoard, DSP_BUFLEN, Framesize * NumChan) ' buffer size

   dw1 = 0
   dw2 = 0

   ret = DSSetDSPProperty(hBoard, DSP_MAXVAL1, dw1)

   ret = DSSetDSPProperty(hBoard, DSP_MAXVAL2, dw2)

   ret = DSSetDSPProperty(hBoard, DSP_OVERLAP, CLng(txtOverLap.Text))  ' FFT
                                                                       ' overlap

' properties below only required for Spectrum Analyzer operation (mode 5)

   ret = DSSetDSPProperty(hBoard, DSP_FFTLEN, FFTsize)

   ret = DSSetDSPProperty(hBoard, DSP_FFTORD, CLng(Log(FFTsize) / Log(2) + 0.1))

   ret = DSSetDSPProperty(hBoard, DSP_LOGFLG1, CInt(LogMagDisplay))
   ret = DSSetDSPProperty(hBoard, DSP_LOGFLG2, CInt(LogMagDisplay))

   ret = DSSetDSPProperty(hBoard, DSP_PHZREQ, CInt(PhaseDisplay))

   ret = DSSetDSPProperty(hBoard, DSP_PWRFLG1, 0)
   ret = DSSetDSPProperty(hBoard, DSP_PWRFLG2, 0)
```

```
   ret = DSSetDSPProperty(hBoard, DSP_XFERFLG1, 0)
   ret = DSSetDSPProperty(hBoard, DSP_XFERFLG2, 0)
   ret = DSSetDSPProperty(hBoard, DSP_DUPFLG, 0)

   LogCoeffA = 20# * 32767 / 100
   LogCoeffB = 32767# - LogCoeffA * (Log(32767) / Log(10))

   ret = DSIEEEToDSP(hBoard, DS_DTI_IEEESIZE32, LogCoeffA, DSPVal, 1)

   ret = DSSetDSPProperty(hBoard, DSP_LOGCOEFFA, DSPVal)

   ret = DSIEEEToDSP(hBoard, DS_DTI_IEEESIZE32, LogCoeffB, DSPVal, 1)

   ret = DSSetDSPProperty(hBoard, DSP_LOGCOEFFB, DSPVal)
```

Run Board; Run One or More Processors

```
' run board/processor examples

   If (fMultiProcessor) Then
      DSRunProcessor(hBoard, dwProcList)
   ElseIf (fMultiProcessor And fFirstProcActiveOnly) Then
     DSRunProcessor(hBoard, &H01)
   ElseIf (fMultiProcessor And fAllProcActive) Then
     DSRunBoard(hBoard)
   ElseIf (fSingleProcessor) Then
    DSRunBoard(hBoard)
   End If
```

Buffer-Ready Callback Message and I/O Data Buffer Processing

```
Sub Buffer_Ready_Message(ByVal hWnd As Long, ByVal MsgVal As Long, wParam As Long, lParam As
Long, nPassage As Integer, lReturnVal As Long)


Dim uStatus As Integer
Dim ret As Integer


   DoEvents  ‘ needed in Win16 VB3/4 only

   If (MsgVal = WM_DSPENGINE_BUFRDY) Then

      If (Not fRunning) Then  ' board already stopped
          Exit Sub
      End If

      If (hBoard = NUL) Then  ' close is in progres
         Exit Sub
      End If

    ' transfer below illustrates two common type of DSP memory architectures

      If (LinMagDisplay Or LogMagDisplay Or PhaseDisplay) Then

         If (uMemArch = DS_GMA_VECTOR) Then  ' vector data memory

          ' magnitude data always in X: memory

            uStatus = DSGetMem(hBoard, DS_GM_VECTOR_DATA_X,
                               dwMagBufferBaseAddr, DS_GM_SIZE16,
                               FFTBuffer(0), 2 * FFTsize * NumChan)

         Else  ' magnitude data not double-buffered

            uStatus = DSGetMem(hBoard, DS_GM_LINEAR_DATA_RT,
                               dwMagBufferBaseAddr, DS_GM_SIZE16,
                               FFTBuffer(0), 2 * FFTsize * NumChan)
```

```
         End If

      ElseIf (TimeDisplay) Then

         If (uMemArch = DS_GMA_VECTOR) Then  ' vector data memory

            If (nCurBuf = 0) Then

               uStatus = DSGetMem(hBoard, DS_GM_VECTOR_DATA_X,
                                  dwTimBufferBaseAddr, DS_GM_SIZE16,
                                  Buffer(0), Framesize * NumChan)

            Else

               uStatus = DSGetMem(hBoard, DS_GM_VECTOR_DATA_Y,
                                  dwTimBufferBaseAddr, DS_GM_SIZE16,
                                  Buffer(0), Framesize * NumChan)

            End If

         Else  ' linear data/prog memory, or modified harvard arch.
               ' with linear data memory

            uStatus = DSGetMem(hBoard, DS_GM_LINEAR_DATA_RT,
                               dwTimBufferBaseAddr + nCurBuf * Framesize *
                               NumChan, DS_GM_SIZE16, Buffer(0), Framesize *
                               NumChan)

         End If
      End If


      If (uStatus = NUL) Then
         ret = DSMessageBox(Spcana32.hWnd, "DSGetMem:  problem with point
                            transfer", szApp, MB_ICONEXCLAMATION)
      End If

      nCurBuf = nCurBuf Xor 1  ' switch buffers

      ret = DSSetDSPProperty(hBoard, DSP_HOSTBUFNUM, nCurBuf)  ' tell DSP about
                                                                ' buffer switch

    ' instruct engine to wait for next buffer
    ' note this is deferred until after data display to avoid Close Event
    ' problem in VB32

      ret = DSWaitForBuffer(hBoard, nCurBuf, 0&, DS_WFB_POLLED)


    ' example of plotting input data points

      fDataRcvd = True
      If (fFirstBuffer = 0) Then
        fFirstBuffer = 1             ' first buffer received
      Else
        fFirstBuffer = 2             ' successive buffer received
      End If

      DisplayBufferData(Buffer)

   End If

   DoEvents  ‘ Win16 only

End Sub
```

Waveform Data Display

The example below uses Microsoft's graph32.ocx control to show how to display different types of waveform data transferred from DSP memory. Note that multichannel data is not shown, but is easy to incorporate. Multichannel data is interleaved in Number-of-Channel groups; for example, for 4-channel data:

ch0,ch1,ch2,ch3,ch0,ch1,ch2,ch3, ...

```
Private Sub DisplayBufferData(Buffer As Integer*)

Dim i As Long
Dim j As Integer
Dim Data As Integer          ' temporary storage for data in buffer
Dim ret, inc, index As Integer

' update the scope window with data currently in Buffer

   graph_window.NumSets = NumChan
   graph_window.ThisSet = 1
   graph_window.ThisPoint = 1

   VOffset = 0
   Vscale = 1

' note:  current display assumes real-valued input data to FFT;
' spectrum is symmetric so only half of freq. domain data needs to be displayed

   If (LinMagDisplay Or LogMagDisplay Or PhaseDisplay) Then
      graph_window.NumPoints = FFTsize / NumChan / 2
      inc = 2
   Else
      graph_window.NumPoints = Framesize / NumChan
      inc = 1
   End If

   If (PhaseDisplay Or LogMagDisplay) Then
      index = 1
   Else
      index = 0
   End If

   graph_window.ThisSet = 1

   For i = 0 To graph_window.NumPoints - 1

      If (LinMagDisplay Or LogMagDisplay Or PhaseDisplay) Then
         Data = FFTBuffer(index)
      ElseIf (TimeDisplay) Then
         Data = Buffer(index)
      End If

      graph_window.GraphData = Data * VScale + VOffset

      index = index + inc

   Next

   For j = 1 To graph_window.NumSets

      graph_window.ThisSet = j
      graph_window.ColorData = j ' Set different channel to different color
      graph_window.DrawMode = 3  ' Blt to screen, to avoid flicker
   Next

End Sub
```

Cleanup (Board Shutdown)

```
Private Sub Form_Unload(Cancel As Integer)

Dim ret As Long


' free board, close engine if needed

   If (hBoard <> NUL) Then

      If (fBoardInitialized <> 0) Then
         ret = DSDisableBoard(hBoard)
      End If

      ret = DSFreeBoard(hBoard)
      hBoard = NUL

   End If

   If (hEngine <> NUL) Then
      ret = DSEngineClose(hEngine)
      hEngine = NUL
   End If

   End  ' shut-down application

End Sub
```

5.1.5 C54xx Source Code Interface Structure

Working with DSP Source Code Interface software for Hypersignal is very straightforward. The fact that there exists similar implementations for nine (9) different DSP chip families has led to standardization of library calls and operational procedures, from the function level down to the batch files used to facilitate the code modification process. For example, all DSP chip families have similar routines available for post-FFT processing, such as log, square root, arctan, etc. And, at a lower level, all "LNK.BAT" files produce a map file of the form "xxxLINK.MAP" where xxx is “T54” for Texas Instruments C54xx, "T30" for Texas Instruments C3x, "M56" for Motorola 56xxx, “A60” for Analog Devices 2106x, etc. Moreover, the modification and download process is also standardized between DSP families. For example, Hypersignal parses and downloads output files from each DSP manufacturer's linker to the DSP board for subsequent execution, obviating the need for user-written object file downloaders.

All interrupt service routines fully save and restore machine context to protect registers they modify and some routines (such as the “snap-in” real-time filter routines included in the C54xx source code library also save and restore registers they use. In all chip families the Digital Oscilloscope and Spectrum Analyzer functions provide the best examples of full context-save interrupt service routines combined with substantial foreground processing constructed from many of the routines in the source code library.

The C54xx Source Code Interface base file list includes .asm and .c function and analog I/O files plus linker control files and support utilities.

The C54xx source code interface supports the basic "modes" of operation below:

C54xx .Asm File	Function	operating mode
MODUL0	Analog Conversion	mode=0
MODUL1	Analog Conversion, Playback from Waveform Display	mode=1
MODUL2	Digital Oscilloscope	mode=2
MODUL3	FFT file-based acceleration	mode=3
MODUL4	iFFT file-based acceleration	mode=4
MODUL5	Spectrum Analyzer	mode=5
MODUL6	Stimulus & Response operation	mode=6
MODUL7	Convolution/Correlation file-based acceleration	mode=7
MODULn	User-Defined function	mode=n

5.1.5.1 C54xx Source Code Interface Interrupt Service Routines

There are various interrupt service routines that respond to interrupts on the C54xx boards and generally match up with the foreground processing modules above. Interrupt service routine module names consist of "ISRn", where n indicates the operating mode (see section 5.1.5 above) that they support, for example "ISR0". Inside each interrupt service routine there are different entry points that contain hardware-specific input/output code for individual boards. These hardware-specific sections are kept as small as possible, so that the great majority of the routine is common for all boards. Examine the interrupt vector and mask initialization in the initial setup code in "T54BEG", depending on board type (see host-shared variable "BOARD"), to locate the correct interrupt service routine for a particular function.

5.1.5.2 C54xx Source Code Interface Host PC Communication

It is important to keep in mind the resident, interrupt-driven interface between host software and the C54xx code. A very small “talker” program is used on the DSP-side to allow two-way communication between the host PC and C54xx DSP to be established. Certain scratch memory locations used by the talker, and the program code area itself used by the talker, absolutely must not be changed in order to maintain high-level host software communication. For example, application-specific DSP code should be linked such that new code does not “step on” the talker;

one way to do this is to leave an non-initialized section, or hole, in the linker command file. An example linker command file is printed below which shows how to do this.

In addition, there are several pre-defined “DSP properties” which can be set by the host to initialize and control default operating modes. These properties are implemented as shared, global variables in the DSP code; they are well documented in the file t54beg.asm. *Do not insert any code of any type before these addresses that would cause them to move*.

The talker currently resides in onchip program memory, from 0x80 to 0x100. In addition, it uses 6 words in onchip memory at 0x1000 for a “command structure” used to communicate with host drivers.

See Appendix B, C54xx Talker Program, for talker source code and map file.

5.1.5.3 C54xx Source Code Interface Buffers and Real-Time Streaming

Currently, operating mode 5 (Spectrum Analyzer) is the best example of an extensive foreground processing combined with context-save background interrupts. The Spectrum Analyzer source code represents a totally real-time system, with samples buffered in a dual-buffer scheme in response to background "sample ready" interrupts, and buffers, when full, sent to FFT and post-FFT processing in the foreground. Use mode 5 source modules ("MODUL5" and modules it calls), and "ISR5" for one example of a real-time, interrupt-driven structure.

5.1.6 C54xx Source Code File List

A partial list of source code and related files included in the C54xx DSP Source Code Interface software follows:

T54DEF.ASM	Variable & constant definition
T54BEG.ASM	Beginning vectors, including ISR vectors, and including initialization, including PC-written variables
ISR0.ASM	Continuous A/D ISR for all board
ISR1.ASM	Continuous D/A ISR for all boards
ISR2.ASM	Digital Scope ISR for all boards.
ISR5.ASM	Spectrum Analyzer ISR for all boards.
ISR6.ASM	Stimulus & Response ISR for all boards.
MODUL0.ASM	Foreground A/D-to-disk processing (pre-defined mode=0)
MODUL1.ASM	Foreground D/A-from-disk processing (pre-defined mode=1)

CONVLV.ASM	Convolution/correlation routine
DIVIDE.ASM	Fractional and integer divide routines
EXP.ASM	Exponentiation (base-2) routine (needed for fixed-point transfer function calculation)
FFT.ASM	Complex FFT/iFFT routine of variable length N ($N=2^m$ where m is FFT order)
FFTOUT.ASM	Post-FFT processing (magnitude, phase, scaling)
FILTER.ASM	Invokes correct filter for current channel
FIR.ASM	FIR filter routine
IIR2.ASM	IIR filter cascade structure 2 (modified direct-form)
IIR3.ASM	IIR filter cascade structure 3 (balanced pole-zero)
LOG.ASM	Log2 approximation routine
MAG.ASM	Magnitude routine
MAXFIN.ASM	Find maximum routine
MOVE.ASM	Move data
PHASE.ASM	4-quadrant phase routine ($-\pi$ to $+\pi$, complex input)
SINTAB.ASM	Sine value lookup-table for FFT use
SPAOUT.ASM	Post-FFT processing for Spectrum Analyzer function, including magnitude, phase, power-spectra, log, scaling, and data manipulation
SQRT.ASM	Square-root routine
USERPROC.C	Example UserProc entry-point function
WINDOW.ASM	Apply pre-defined window (used with FFTs)
XFER.ASM	Complex-ratio routine used for transfer function
ASM.BAT	Batch file used to assemble any source module: type "ASM filename" (no extension needed)
BUILD.BAT	Batch file used to assemble and compile all .asm and .c files

LNK.BAT	Batch file used to link all source modules: type "LNK"
T54LINK.CMD	Link control file (batch file for linker)
AD50INIT.ASM	Initialize TLC320AD50 codec and McBSP1
BSP.C	BSP / McBSP initialization
C_INIT.ASM	Initialize C code run-time variables
DSK_INIT.ASM	Initialize AIC TLC320AC01 codec and TDM port for old-style DSK C54x board
FAR_MODE.MAC	Macros used to build far-call, far-return .asm files
ISR_5402.ASM	ISR entry-point for DSK C5402 board analog I/O
ISR_DSK542.ASM	ISR entry-point for old-style DSK C542 board analog I/O
ISR_SD4.ASM	ISR entry-point for Signalogic SD4 Audio Module
SD4_INT.C	SigSD4 Audio Module initialization
SS_INIT.C	Synchronous serial initialization
STANDALONE.C	Stand-Alone Mode Initialization Function
C54XX.H	Basic C54xx processor and C code definitions
RTC_C54XX.H	Property alias definitions allow linking to host-shared properties defined in t54def.asm
BSP.H	Definitions for BSP and McBSP ports on C54xx and C62xx devices
C54XX_SCI.PJT	C54xx source code interface project file
TMSC54X.MAP	Map file for C54x Source Code Interface

```
******************************************************************************
TMS320C54x COFF Linker           PC Version 3.70
******************************************************************************
>> Linked Wed Jul 24 19:12:30 2002

OUTPUT FILE NAME:   <./tmsc54x.out>
ENTRY POINT SYMBOL: "_c_int00"  address: 00000401


MEMORY CONFIGURATION

                  name            origin    length      used    attr    fill
         ----------------------  --------  ---------  --------  ----  --------
PAGE  0: PROG                    00000080   00007d80  00001d52  RWIX

PAGE  1: DATA                    00000300   00000080  00000003  RWIX


SECTION ALLOCATION MAP
```

```
 output                                  attributes/
section   page    origin      length       input sections
--------  ----  ----------  ----------   ----------------
RESET       0    00000080    00000070
                   00000080    00000070     t54beg.obj (RESET)

HI_START    0    00000100    00000080     UNINITIALIZED

PCDATA      0    00000200    00000100     UNINITIALIZED
                   00000300    00000000     T54def.obj (PCDATA)

DATA        0    00000300    00000080     UNINITIALIZED

FILT        0    00000380    00000080     UNINITIALIZED

INIT        0    00000400    00000015
                   00000400    00000015     t54beg.obj (INIT)

MEMINI      0    00000500    00000275
                   00000500    00000275     t54beg.obj (MEMINI)

VARS        0    0000042f    0000005e
                   0000042f    0000005e     T54def.obj (VARS)

DPRAM       0    00001000    00000100     UNINITIALIZED

.stack      0    00001100    00000100     UNINITIALIZED
                   00001100    00000000     c_init.obj (.stack)

SINTAB      0    00001200    00000600
                   00001200    00000600     Sintab.obj (SINTAB)

.bss        1    00000300    00000003     UNINITIALIZED
                   00000300    00000000     AD50Init.obj (.bss)
                   00000300    00000000     Xfer.obj (.bss)
                   00000300    00000000     Window.obj (.bss)
                   00000300    00000000     userproc.obj (.bss)
                   00000300    00000000     T54def.obj (.bss)
                   00000300    00000000     t54beg.obj (.bss)
                   00000300    00000000     standalone.obj (.bss)
                   00000300    00000000     ss_init.obj (.bss)
                   00000300    00000000     Sqrt.obj (.bss)
                   00000300    00000000     Spaout.obj (.bss)
                   00000300    00000000     Sintab.obj (.bss)
                   00000300    00000000     phase.obj (.bss)
                   00000300    00000000     Move.obj (.bss)
                   00000300    00000000     Modul7.obj (.bss)
                   00000300    00000000     Modul6.obj (.bss)
                   00000300    00000000     Modul5.obj (.bss)
                   00000300    00000000     Modul4.obj (.bss)
                   00000300    00000000     Modul3.obj (.bss)
                   00000300    00000000     Modul2.obj (.bss)
                   00000300    00000000     Modul1.obj (.bss)
                   00000300    00000000     Modul0.obj (.bss)
                   00000300    00000000     Maxfin.obj (.bss)
                   00000300    00000000     Mag.obj (.bss)
                   00000300    00000000     Log.obj (.bss)
                   00000300    00000000     isr_sd4.obj (.bss)
                   00000300    00000000     isr_dsk542.obj (.bss)
                   00000300    00000000     isr_5402.obj (.bss)
                   00000300    00000000     Isr6.obj (.bss)
                   00000300    00000000     Isr5.obj (.bss)
                   00000300    00000000     Isr2.obj (.bss)
                   00000300    00000000     Isr1.obj (.bss)
                   00000300    00000000     Isr0.obj (.bss)
                   00000300    00000000     iir3.obj (.bss)
                   00000300    00000000     Iir2.obj (.bss)
                   00000300    00000000     Fir.obj (.bss)
                   00000300    00000000     Filter.obj (.bss)
                   00000300    00000000     Fftout.obj (.bss)
                   00000300    00000000     Fft.obj (.bss)
```

```
                  00000300    00000000     Exp.obj (.bss)
                  00000300    00000000     dsk_init.obj (.bss)
                  00000300    00000000     divide.obj (.bss)
                  00000300    00000000     Convlv.obj (.bss)
                  00000300    00000000     c_init.obj (.bss)
                  00000300    00000000     bsp.obj (.bss)
                  00000300    00000000     arctan.obj (.bss)
                  00000300    00000003     sd4_init.obj (.bss)

.text      0    00000775    000005e9
                  00000775    0000008d     AD50Init.obj (.text)
                  00000802    00000000     Xfer.obj (.text)
                  00000802    00000000     Window.obj (.text)
                  00000802    00000000     T54def.obj (.text)
                  00000802    00000000     t54beg.obj (.text)
                  00000802    00000000     Sqrt.obj (.text)
                  00000802    00000000     Spaout.obj (.text)
                  00000802    00000000     Sintab.obj (.text)
                  00000802    00000000     phase.obj (.text)
                  00000802    00000000     Move.obj (.text)
                  00000802    00000000     Modul7.obj (.text)
                  00000802    00000000     Modul5.obj (.text)
                  00000802    00000000     Modul4.obj (.text)
                  00000802    00000000     Modul3.obj (.text)
                  00000802    00000000     Modul2.obj (.text)
                  00000802    00000000     Modul1.obj (.text)
                  00000802    00000000     Modul0.obj (.text)
                  00000802    00000000     Maxfin.obj (.text)
                  00000802    00000000     Mag.obj (.text)
                  00000802    00000000     Log.obj (.text)
                  00000802    00000000     isr_sd4.obj (.text)
                  00000802    00000000     isr_dsk542.obj (.text)
                  00000802    00000000     isr_5402.obj (.text)
                  00000802    00000000     Isr6.obj (.text)
                  00000802    00000000     Isr5.obj (.text)
                  00000802    00000000     Isr2.obj (.text)
                  00000802    00000000     Isr1.obj (.text)
                  00000802    00000000     Isr0.obj (.text)
                  00000802    00000000     iir3.obj (.text)
                  00000802    00000000     Iir2.obj (.text)
                  00000802    00000000     Fir.obj (.text)
                  00000802    00000000     Filter.obj (.text)
                  00000802    00000000     Fftout.obj (.text)
                  00000802    00000000     Fft.obj (.text)
                  00000802    00000000     Exp.obj (.text)
                  00000802    00000000     divide.obj (.text)
                  00000802    00000000     Convlv.obj (.text)
                  00000802    00000000     arctan.obj (.text)
                  00000802    000002f3     bsp.obj (.text)
                  00000af5    00000036     c_init.obj (.text)
                  00000b2b    00000066     dsk_init.obj (.text)
                  00000b91    00000031     Modul6.obj (.text)
                  00000bc2    000000ca     sd4_init.obj (.text)
                  00000c8c    0000003f     ss_init.obj (.text)
                  00000ccb    00000069     standalone.obj (.text)
                  00000d34    0000002a     userproc.obj (.text)

.data      1    00000000    00000000     UNINITIALIZED
                  00000000    00000000     AD50Init.obj (.data)
                  00000000    00000000     Xfer.obj (.data)
                  00000000    00000000     Window.obj (.data)
                  00000000    00000000     userproc.obj (.data)
                  00000000    00000000     T54def.obj (.data)
                  00000000    00000000     t54beg.obj (.data)
                  00000000    00000000     standalone.obj (.data)
                  00000000    00000000     ss_init.obj (.data)
                  00000000    00000000     Sqrt.obj (.data)
                  00000000    00000000     Spaout.obj (.data)
                  00000000    00000000     Sintab.obj (.data)
                  00000000    00000000     sd4_init.obj (.data)
                  00000000    00000000     phase.obj (.data)
```

```
                  00000000    00000000     Move.obj (.data)
                  00000000    00000000     Modul7.obj (.data)
                  00000000    00000000     Modul6.obj (.data)
                  00000000    00000000     Modul5.obj (.data)
                  00000000    00000000     Modul4.obj (.data)
                  00000000    00000000     Modul3.obj (.data)
                  00000000    00000000     Modul2.obj (.data)
                  00000000    00000000     Modul1.obj (.data)
                  00000000    00000000     Modul0.obj (.data)
                  00000000    00000000     Maxfin.obj (.data)
                  00000000    00000000     Mag.obj (.data)
                  00000000    00000000     Log.obj (.data)
                  00000000    00000000     isr_sd4.obj (.data)
                  00000000    00000000     isr_dsk542.obj (.data)
                  00000000    00000000     isr_5402.obj (.data)
                  00000000    00000000     Isr6.obj (.data)
                  00000000    00000000     Isr5.obj (.data)
                  00000000    00000000     Isr2.obj (.data)
                  00000000    00000000     Isr1.obj (.data)
                  00000000    00000000     Isr0.obj (.data)
                  00000000    00000000     iir3.obj (.data)
                  00000000    00000000     Iir2.obj (.data)
                  00000000    00000000     Fir.obj (.data)
                  00000000    00000000     Filter.obj (.data)
                  00000000    00000000     Fftout.obj (.data)
                  00000000    00000000     Fft.obj (.data)
                  00000000    00000000     Exp.obj (.data)
                  00000000    00000000     dsk_init.obj (.data)
                  00000000    00000000     divide.obj (.data)
                  00000000    00000000     Convlv.obj (.data)
                  00000000    00000000     c_init.obj (.data)
                  00000000    00000000     bsp.obj (.data)
                  00000000    00000000     arctan.obj (.data)

ARCTAN      0     00000180    0000001e
                  00000180    0000001e     arctan.obj (ARCTAN)

CONVLV      0     0000019e    0000005d
                  0000019e    0000005d     Convlv.obj (CONVLV)

DIVIDE      0     0000048d    00000036
                  0000048d    00000036     divide.obj (DIVIDE)

EXP         0     00000d5e    00000051
                  00000d5e    00000051     Exp.obj (EXP)

FFT         0     00000daf    000000be
                  00000daf    000000be     Fft.obj (FFT)

FFTOUT      0     000004c3    00000030
                  000004c3    00000030     Fftout.obj (FFTOUT)

FILTER      0     00000e6d    00000064
                  00000e6d    00000064     Filter.obj (FILTER)

FIR         0     00000415    00000015
                  00000415    00000015     Fir.obj (FIR)

IIR2        0     00000ed1    00000014
                  00000ed1    00000014     Iir2.obj (IIR2)

IIR3        0     00000ee5    0000001a
                  00000ee5    0000001a     iir3.obj (IIR3)

ISR0        0     00000eff    00000084
                  00000eff    00000084     Isr0.obj (ISR0)

ISR1        0     00000f83    00000045
                  00000f83    00000045     Isr1.obj (ISR1)

ISR2        0     00001800    00000081
```

```
                  00001800    00000081     Isr2.obj (ISR2)

ISR5        0     00001881    000000a0
                  00001881    000000a0     Isr5.obj (ISR5)

ISR6        0     00001921    000000b8
                  00001921    000000b8     Isr6.obj (ISR6)

ISR_5402    0     00000fc8    0000001a
                  00000fc8    0000001a     isr_5402.obj (ISR_5402)

ISR_DSK542    0     00000fe2    00000014
                  00000fe2    00000014     isr_dsk542.obj (ISR_DSK542)

isr_code    0     000019d9    00000080
                  000019d9    00000080     isr_sd4.obj (isr_code)

LOG         0     00001a59    00000038
                  00001a59    00000038     Log.obj (LOG)

MAG         0     000000f0    00000006
                  000000f0    00000006     Mag.obj (MAG)

MAXFIN      0     00001a91    0000001b
                  00001a91    0000001b     Maxfin.obj (MAXFIN)

MODUL0      0     00001aac    00000012
                  00001aac    00000012     Modul0.obj (MODUL0)

MODUL1      0     000004f3    0000000d
                  000004f3    0000000d     Modul1.obj (MODUL1)

MODUL2      0     00001abe    00000013
                  00001abe    00000013     Modul2.obj (MODUL2)

MODUL3      0     00001ad1    00000032
                  00001ad1    00000032     Modul3.obj (MODUL3)

MODUL4      0     00001b03    00000020
                  00001b03    00000020     Modul4.obj (MODUL4)

MODUL5      0     00001b23    00000083
                  00001b23    00000083     Modul5.obj (MODUL5)

MODUL7      0     00001ba6    0000006f
                  00001ba6    0000006f     Modul7.obj (MODUL7)

MOVE        0     00001c15    00000030
                  00001c15    00000030     Move.obj (MOVE)

PHASE       0     00001c45    00000042
                  00001c45    00000042     phase.obj (PHASE)

SPAOUT      0     00001c87    00000085
                  00001c87    00000085     Spaout.obj (SPAOUT)

SQRT        0     00001d0c    00000029
                  00001d0c    00000029     Sqrt.obj (SQRT)

DEBUG       0     00001d35    00000010
                  00001d35    00000010     t54beg.obj (DEBUG)

WINDOW      0     00001d45    00000042
                  00001d45    00000042     Window.obj (WINDOW)

XFER        0     00001d87    00000069
                  00001d87    00000069     Xfer.obj (XFER)


GLOBAL SYMBOLS: SORTED ALPHABETICALLY BY Name
```

```
address    name
--------   ----
00000300   .bss
00000000   .data
00000775   .text
00000466   ACTADD
00000777   AD50Init
00000237   ADCTR
00000463   AICADD
00000234   AICWRD
00000440   AMPADD
00000211   AMPSCL
0000045d   ANAADD
0000022e   ANABFK
0000025c   ANGLE
00000448   AP2ADD
00000455   APCADD
00000226   APCOEF
00000219   APSCL2
00000180   ARCTAN
0000025b   BASANG
0000042f   BDADD
00000469   BF2ADD
0000025a   BFDONE
0000043d   BFLADD
0000043f   BFNADD
00000200   BOARD
00000456   BPCADD
00000227   BPCOEF
0000020e   BUFLEN
00000210   BUFNUM
0000024b   BUFPTR
00000431   BZYADD
00000202   BZYFLG
00000445   CF1ADD
0000044a   CF2ADD
00000446   CHLADD
00000217   CHNLIST
000003c0   COEFF1
000003e0   COEFF2
00000216   COFXF1
0000021b   COFXF2
0000019e   CONVLV
00000249   COUNT
0000020a   COUPL
00000439   CPLADD
00000243   CURCHN
00000259   CURTRC
00000238   DACTR
00000467   DCTADD
00000380   DELAY1
000003a0   DELAY2
0000025d   DENOM
00000449   DF2ADD
0000021a   DFSET2
00000241   DISCARD
00000441   DOFADD
00000212   DOFSET
00000261   DSIGN
00000b2b   DSKInit
00000200   DSOFF
00000004   DSPAGE
00000444   DUPADD
00000215   DUPCHN
00000d5e   EXP2
000004a3   FDIV
00000264   FDRVAL
00000daf   FFT
00000436   FFTADD
00000255   FFTBUF
00000254   FFTFLG
```

```
000004c3   FFTOUT
00000207   FFTSIZ
00000e6d   FILTER
0000023d   FILUPD
00000415   FIR
0000044d   FL1ADD
0000044e   FL2ADD
0000021e   FLLEN1
0000021f   FLLEN2
00000222   FLTYP1
0000023b   FLTYP2
00000206   FRMSIZ
00000426   FS1
0000043b   FSMADD
0000020c   FS_MODE
00000223   FS_VAL
00000451   FT1ADD
0000046a   FT2ADD
0000046c   FUPADD
0000043a   GNLADD
0000020b   GNLIST
0000026b   I
00000ed5   I2L1
00000edb   I2S1
00000ee2   I2S2
00000eea   I3L1
00000eed   I3L2
00000ef0   I3L3
00000efd   I3L4
00000eef   I3S1
00000ef4   I3S2
00000ee8   I3S3
0000026f   IA
00000275   IADD
0000048d   IDIV
00000270   IE
00000ed1   IIR2
00000ee5   IIR3
00000278   IMAG
0000021d   IMAX
0000044c   IMXADD
00000747   INT_ENABLE
00000eff   ISR0_COMMON
00000f83   ISR1_COMMON
00001800   ISR2_COMMON
00001881   ISR5_COMMON
00001921   ISR6_COMMON
00000fc8   ISR_5402
00000fd6   ISR_5402_JUMP
00000fe2   ISR_DSK542
00000ff4   ISR_DSK542_JUMP
00001a4a   ISR_SD4_JUMP
000019d9   ISR_SD4_RX
00001a10   ISR_SD4_TX
00000274   J
0000026c   L
00000276   LADD
00000464   LC1ADD
00000465   LC2ADD
0000044f   LG1ADD
00000450   LG2ADD
00000220   LGFLG1
00000221   LGFLG2
00001a59   LOG2
00000235   LOGC1
00000236   LOGC2
00000272   LOGFLG
000000f0   MAG
0000027b   MAGVAL
0000022a   MAXA
00000437   MAXADD
```

```
0000022b   MAXB
0000000b   MAXFFTSIZE
00001a91   MAXFIN
00000279   MAXI
00000208   MAXMAG
00000462   MDLADD
00000433   MINADD
0000027a   MINI
00000204   MINMAG
00000430   MODADD
00001aac   MODUL0
000004f3   MODUL1
00001abe   MODUL2
00001ad1   MODUL3
00001b03   MODUL4
00001b23   MODUL5
00001ba6   MODUL7
00000233   MODULE
00000201   MODVAL
00000461   MONADD
00000232   MONTOR
00001c15   MOVE
0000045f   MPBADD
00000230   MPBBFK
00000258   MPBSAV
00000459   MXAADD
0000045a   MXBADD
00000272   N1
00000273   N2
0000045b   NCHADD
0000027c   NSTAT
0000022c   NUMCHN
0000024e   NUMCHNM1
0000025e   NUMER
0000023f   ONE
00000434   ORDADD
00000205   ORDVAL
00000252   OUTPC
00000246   OUTTRA
00000248   OUTTRB
00000432   OVFADD
00000203   OVFFLG
0000043e   PCBADD
0000020f   PCBFLG
00001c45   PHASE
00000443   PHRADD
00000214   PHZREQ
00006488   PI
00003244   PIOV2
0000024d   PRVTRG
00000453   PW1ADD
00000454   PW2ADD
00000224   PWFLG1
00000225   PWFLG2
00000273   PWRFLG
0000046b   QNTADD
0000023c   QNUM
00000271   QUARTN
0000025f   QUOTNT
00000277   REAL
00000256   RIB1
00000257   RIB2
0000045e   RIBADD
0000022f   RIBBFK
00000438   RIFADD
00000209   RIFLAG
0000021c   RMAX
0000044b   RMXADD
00000251   SAMP1
00000245   SAMPLA
00000247   SAMPLB
```

```
00000244   SAMPLE
0000045c   SBTADD
00001200   SINTAB
00000435   SIZADD
00001c87   SPAOUT
0000024a   SPTEMP
00001d0c   SQRT
00000452   SR2ADD
0000023a   STBFLN
0000025b   STBPTR
00000dce   STFOR1
00000dd1   STFOR2
00000468   STMADD
00000239   STMBFK
0000022d   SUBTYP
00000253   SYNFLG
00000275   TASAV
00000276   TBSAV
00000447   TCHADD
0000024f   TEMP
00000250   TEMP2
00000260   TEMPD1
00000457   TF1ADD
00000458   TF2ADD
00000228   TFFLG1
00000229   TFFLG2
00000274   TFRFLG
00000245   TRACE1
00000247   TRACE2
0000043c   TRGADD
00000218   TRGCHN
0000024c   TRGFLG
0000020d   TRGVAL
0000026e   WI
00000442   WINADD
00001d45   WINDOW
00000213   WINSCL
00000460   WNDADD
00000231   WNDBFK
0000026d   WR
00001d87   XFER
00000265   XI
00000267   XL
00000269   XT
00000266   YI
00000268   YL
0000026a   YT
00000240   ZERO
00000211   _AMPSCL
0000022e   _ANABFK
00000219   _APSCL2
0000043f   _BFNADD
00000200   _BOARD
0000020e   _BUFLEN
00000210   _BUFNUM
0000024b   _BUFPTR
00000217   _CHNLIST
00000216   _COFXF1
0000021b   _COFXF2
00000302   _ChanList_Shifted
0000021a   _DFSET2
00000212   _DOFSET
0000023d   _FILUPD
0000021e   _FLLEN1
0000021f   _FLLEN2
00000222   _FLTYP1
0000023b   _FLTYP2
00000206   _FRMSIZ
0000020c   _FS_MODE
00000223   _FS_VAL
00000ab4   _InitBSP
```

```
00000954   _InitMcBSP
00000747   _InterruptEnable
0000022a   _MAXA
0000022b   _MAXB
00000b91   _MODUL6
00000201   _MODVAL
00000230   _MPBBFK
00000459   _MXAADD
0000045a   _MXBADD
0000022c   _NUMCHN
00000246   _OUTTRA
0000043e   _PCBADD
0000020f   _PCBFLG
0000027d   _ProcessorType
0000023c   _QNUM
0000022f   _RIBBFK
00000301   _RxByteFlag
000019fd   _RxRead1
00001a03   _RxRead2
00001a0c   _RxRead3
00000bc2   _SD4Init
0000023a   _STBFLN
0000025b   _STBPTR
00000239   _STMBFK
0000022d   _SUBTYP
00000ccb   _StandAloneInit
00000c8c   _SyncSerialInit
00000300   _TxByteFlag
00001a37   _TxWrite1
00001a3c   _TxWrite2
00001a41   _TxWrite3
00000d34   _UserProc
00000231   _WNDBFK
00000100   __STACK_SIZE
00000300   ___bss__
ffffffff   ___cinit__
00000000   ___data__
00000000   ___edata__
00000303   ___end__
00000d5e   ___etext__
ffffffff   ___pinit__
00000775   ___text__
00000001   __lflags
00000af5   _c_init
00000401   _c_int00
00000802   _getMcBSPInfo
0000088f   _getMcBSPSubReg
000008a3   _setMcBSPSubReg
000008b8   _setMcBSPSubRegAll
ffffffff   cinit
00000000   edata
00000303   end
00000d5e   etext
ffffffff   pinit


GLOBAL SYMBOLS: SORTED BY Symbol Address

address    name
--------   ----
00000000   ___edata__
00000000   ___data__
00000000   edata
00000000   .data
00000001   __lflags
00000004   DSPAGE
0000000b   MAXFFTSIZE
000000f0   MAG
00000100   __STACK_SIZE
00000180   ARCTAN
0000019e   CONVLV
```

```
00000200   BOARD
00000200   _BOARD
00000200   DSOFF
00000201   _MODVAL
00000201   MODVAL
00000202   BZYFLG
00000203   OVFFLG
00000204   MINMAG
00000205   ORDVAL
00000206   _FRMSIZ
00000206   FRMSIZ
00000207   FFTSIZ
00000208   MAXMAG
00000209   RIFLAG
0000020a   COUPL
0000020b   GNLIST
0000020c   FS_MODE
0000020c   _FS_MODE
0000020d   TRGVAL
0000020e   _BUFLEN
0000020e   BUFLEN
0000020f   _PCBFLG
0000020f   PCBFLG
00000210   _BUFNUM
00000210   BUFNUM
00000211   AMPSCL
00000211   _AMPSCL
00000212   DOFSET
00000212   _DOFSET
00000213   WINSCL
00000214   PHZREQ
00000215   DUPCHN
00000216   _COFXF1
00000216   COFXF1
00000217   _CHNLIST
00000217   CHNLIST
00000218   TRGCHN
00000219   _APSCL2
00000219   APSCL2
0000021a   DFSET2
0000021a   _DFSET2
0000021b   _COFXF2
0000021b   COFXF2
0000021c   RMAX
0000021d   IMAX
0000021e   _FLLEN1
0000021e   FLLEN1
0000021f   _FLLEN2
0000021f   FLLEN2
00000220   LGFLG1
00000221   LGFLG2
00000222   FLTYP1
00000222   _FLTYP1
00000223   _FS_VAL
00000223   FS_VAL
00000224   PWFLG1
00000225   PWFLG2
00000226   APCOEF
00000227   BPCOEF
00000228   TFFLG1
00000229   TFFLG2
0000022a   _MAXA
0000022a   MAXA
0000022b   MAXB
0000022b   _MAXB
0000022c   NUMCHN
0000022c   _NUMCHN
0000022d   SUBTYP
0000022d   _SUBTYP
0000022e   _ANABFK
0000022e   ANABFK
```

```
0000022f   RIBBFK
0000022f   _RIBBFK
00000230   MPBBFK
00000230   _MPBBFK
00000231   WNDBFK
00000231   _WNDBFK
00000232   MONTOR
00000233   MODULE
00000234   AICWRD
00000235   LOGC1
00000236   LOGC2
00000237   ADCTR
00000238   DACTR
00000239   _STMBFK
00000239   STMBFK
0000023a   _STBFLN
0000023a   STBFLN
0000023b   FLTYP2
0000023b   _FLTYP2
0000023c   QNUM
0000023c   _QNUM
0000023d   FILUPD
0000023d   _FILUPD
0000023f   ONE
00000240   ZERO
00000241   DISCARD
00000243   CURCHN
00000244   SAMPLE
00000245   TRACE1
00000245   SAMPLA
00000246   OUTTRA
00000246   _OUTTRA
00000247   SAMPLB
00000247   TRACE2
00000248   OUTTRB
00000249   COUNT
0000024a   SPTEMP
0000024b   BUFPTR
0000024b   _BUFPTR
0000024c   TRGFLG
0000024d   PRVTRG
0000024e   NUMCHNM1
0000024f   TEMP
00000250   TEMP2
00000251   SAMP1
00000252   OUTPC
00000253   SYNFLG
00000254   FFTFLG
00000255   FFTBUF
00000256   RIB1
00000257   RIB2
00000258   MPBSAV
00000259   CURTRC
0000025a   BFDONE
0000025b   BASANG
0000025b   _STBPTR
0000025b   STBPTR
0000025c   ANGLE
0000025d   DENOM
0000025e   NUMER
0000025f   QUOTNT
00000260   TEMPD1
00000261   DSIGN
00000264   FDRVAL
00000265   XI
00000266   YI
00000267   XL
00000268   YL
00000269   XT
0000026a   YT
0000026b   I
```

```
0000026c   L
0000026d   WR
0000026e   WI
0000026f   IA
00000270   IE
00000271   QUARTN
00000272   N1
00000272   LOGFLG
00000273   N2
00000273   PWRFLG
00000274   J
00000274   TFRFLG
00000275   TASAV
00000275   IADD
00000276   LADD
00000276   TBSAV
00000277   REAL
00000278   IMAG
00000279   MAXI
0000027a   MINI
0000027b   MAGVAL
0000027c   NSTAT
0000027d   _ProcessorType
00000300   _TxByteFlag
00000300   .bss
00000300   ___bss__
00000301   _RxByteFlag
00000302   _ChanList_Shifted
00000303   end
00000303   ___end__
00000380   DELAY1
000003a0   DELAY2
000003c0   COEFF1
000003e0   COEFF2
00000401   _c_int00
00000415   FIR
00000426   FS1
0000042f   BDADD
00000430   MODADD
00000431   BZYADD
00000432   OVFADD
00000433   MINADD
00000434   ORDADD
00000435   SIZADD
00000436   FFTADD
00000437   MAXADD
00000438   RIFADD
00000439   CPLADD
0000043a   GNLADD
0000043b   FSMADD
0000043c   TRGADD
0000043d   BFLADD
0000043e   PCBADD
0000043e   _PCBADD
0000043f   BFNADD
0000043f   _BFNADD
00000440   AMPADD
00000441   DOFADD
00000442   WINADD
00000443   PHRADD
00000444   DUPADD
00000445   CF1ADD
00000446   CHLADD
00000447   TCHADD
00000448   AP2ADD
00000449   DF2ADD
0000044a   CF2ADD
0000044b   RMXADD
0000044c   IMXADD
0000044d   FL1ADD
0000044e   FL2ADD
```

```
0000044f   LG1ADD
00000450   LG2ADD
00000451   FT1ADD
00000452   SR2ADD
00000453   PW1ADD
00000454   PW2ADD
00000455   APCADD
00000456   BPCADD
00000457   TF1ADD
00000458   TF2ADD
00000459   MXAADD
00000459   _MXAADD
0000045a   MXBADD
0000045a   _MXBADD
0000045b   NCHADD
0000045c   SBTADD
0000045d   ANAADD
0000045e   RIBADD
0000045f   MPBADD
00000460   WNDADD
00000461   MONADD
00000462   MDLADD
00000463   AICADD
00000464   LC1ADD
00000465   LC2ADD
00000466   ACTADD
00000467   DCTADD
00000468   STMADD
00000469   BF2ADD
0000046a   FT2ADD
0000046b   QNTADD
0000046c   FUPADD
0000048d   IDIV
000004a3   FDIV
000004c3   FFTOUT
000004f3   MODUL1
00000747   INT_ENABLE
00000747   _InterruptEnable
00000775   .text
00000775   ___text__
00000777   AD50Init
00000802   _getMcBSPInfo
0000088f   _getMcBSPSubReg
000008a3   _setMcBSPSubReg
000008b8   _setMcBSPSubRegAll
00000954   _InitMcBSP
00000ab4   _InitBSP
00000af5   _c_init
00000b2b   DSKInit
00000b91   _MODUL6
00000bc2   _SD4Init
00000c8c   _SyncSerialInit
00000ccb   _StandAloneInit
00000d34   _UserProc
00000d5e   etext
00000d5e   ___etext__
00000d5e   EXP2
00000daf   FFT
00000dce   STFOR1
00000dd1   STFOR2
00000e6d   FILTER
00000ed1   IIR2
00000ed5   I2L1
00000edb   I2S1
00000ee2   I2S2
00000ee5   IIR3
00000ee8   I3S3
00000eea   I3L1
00000eed   I3L2
00000eef   I3S1
00000ef0   I3L3
```

```
00000ef4   I3S2
00000efd   I3L4
00000eff   ISR0_COMMON
00000f83   ISR1_COMMON
00000fc8   ISR_5402
00000fd6   ISR_5402_JUMP
00000fe2   ISR_DSK542
00000ff4   ISR_DSK542_JUMP
00001200   SINTAB
00001800   ISR2_COMMON
00001881   ISR5_COMMON
00001921   ISR6_COMMON
000019d9   ISR_SD4_RX
000019fd   _RxRead1
00001a03   _RxRead2
00001a0c   _RxRead3
00001a10   ISR_SD4_TX
00001a37   _TxWrite1
00001a3c   _TxWrite2
00001a41   _TxWrite3
00001a4a   ISR_SD4_JUMP
00001a59   LOG2
00001a91   MAXFIN
00001aac   MODUL0
00001abe   MODUL2
00001ad1   MODUL3
00001b03   MODUL4
00001b23   MODUL5
00001ba6   MODUL7
00001c15   MOVE
00001c45   PHASE
00001c87   SPAOUT
00001d0c   SQRT
00001d45   WINDOW
00001d87   XFER
00003244   PIOV2
00006488   PI
ffffffff   ___pinit__
ffffffff   pinit
ffffffff   cinit
ffffffff   ___cinit__

[336 symbols]
)
```

The distribution software does not initially contain all .OBJ files necessary to produce a complete, default executable code module when the T54LINK.CMD control file (above) is invoked with the Texas Instruments linker utility. Each .ASM module above must be assembled first; use the BUILD.BAT file to do this.

5.1.7 C54xx Source Code Modification Process

The steps taken to modify, add and integrate new source code are listed below:

1) Make changes to C54xx source code, keeping in mind the necessary shared variables between Hypersignal and the C54xx processor. The file T54DEF.ASM is well documented in this area.

2) Assemble all modules changed and any new modules created (using Texas Instruments "COFF" assembler) to ensure that a new .OBJ (object) file is generated for each altered module. For convenience, an "ASM.BAT" batch file has been provided that can be invoked as

```
ASM filename <car ret>
```

to assemble source code modules (entering the .ASM extension after "filename" is optional). This file executes

```
asm500 -l -v549 -mg %1 > %1.err
type %1.err
```

statements, where %1 is the filename. *If a new module has been created, be sure to add it to the T54LINK.CMD control file* (use an ASCII text-editor to update the T54LINK.CMD file).

3) Re-link all modules using Texas Instruments C54xx "lnk500" linker. A "LNK.BAT" batch file has been provided that can be used by invoking:

```
LNK <car ret>
```

to link all necessary .OBJ files. LNK.BAT executes a

```
lnk500 T54LINK.CMD
```

statement, where "T54LINK.CMD" is an ASCII linker-command file that specifies all .OBJ files to be linked by the dsplnk.exe program, as well as specific information about memory addresses for some sections.

4) Copy the resulting .OUT file to \hsmacro or \dspower subdirectories, run the desired host program, and invoke the appropriate functions. When using DSPower programs, the .OUT filename can be specified in the Hardware Manager dialog box (DSP Program File field) or in a LoadFileProcessor() or LoadFileBoard() function call. When using Hypersignal software, the .OUT filename can be specified in the System Config menu (added after the board designator with a separating comma). It is also possible to simply copy the resulting .OUT file to \hsmacro or \dspower subdirectories over the existing, default tmsc54x.out file. However, in this case, it is suggested that a backup of the original file be made first in case the new code does not work. In either case, the DSPower or Hypersignal COFF file download function calls will read the .OUT file directly according to Texas Instruments COFF file format and download all code and data. A subsequent RunProcessor() or RunBoard() function call will begin execution at the reset vector location.

5) There are several references to "trace" and "channel" in DSP source code and source code variable descriptions. Traces are references to waveforms in displays or waveform files, whereas channels refer to the physical analog input or output connection on the DSP/analog hardware. A

logical-to-physical mapping takes place when channel lists are sent from Hypersignal software or DSPower programs to executing DSP code.

5.1.7.1 C54xx Source Code Interface Modification Notes

Some notes about C54xx source code modification:

1) All code has been tested using version 3.50 of "asm500.exe" and "lnk500.exe".

2) To modify the default time data, frequency domain output data, or other buffer addresses, please consult the comments and suggestions in the T54DEF.ASM file.

3) The LNK.BAT file produces a "T54LINK.MAP" map file which can be used to check for overlaps and current locations of variables, buffers, and code sections. The map file also shows the starting address and length of each code module included in the current executable output file. When significant amounts of code are added, extra attention should be paid to the length of the executable file as compared to the amount of program memory available. If the available program memory is exceeded, problems can occur which may be difficult to pin down. Such problems can seem to result from other causes, and can lead to unnecessary time spent searching for the real cause.

4) The T54LINK.CMD file creates non-initialized "holes" from 0x80 through 0xff and from 0x1000 through 0x1005that is intended to leave room for the “talker” program (see section 5.1.5.2 above). The DSPower and Hypersignal COFF download function calls ignore non-initialized sections in the COFF file and do not download to the area specified by the section.

5) Run-time 'C' support is made necessary by adding C code with functions or routines that require math and other support libraries. Typically; the library file "rtsPP.lib" would be required, where PP is a name assigned by the DSP chip manufacturer (for example “54” for C54xx, "30" for C3x, "56" for 56xxx, etc.). This can be added to the .CMD file being used, or the xxxLINKC.CMD and xxxLINKC.CTL command/control files can be used to link in the appropriate rtsPP.lib file.

6) If DSP program filename has been changed from the default (by renaming the output of lnkc.bat or by giving lnkc.bat a filename parameter), DSPower programs or Hypersignal software must know about the name change. To do this in DSPower programs, change the value of the DSP Program File field in the Hardware Manager dialog box (or change the value of

LoadFileProcessor and LoadFileBoard calls in the programs). To do this in Hypersignal software, suffix the filename to the appropriate field in the System Config menu. For example, if the Hypersignal Analog Conversion function is being used for test, then add the new filename to the ANALOG CONVERSION field in the System Config menu; for example, this might be an entry like 'IIC31-B-50,test.out', 'TBS56-B,AC3.OUT' or 'sig32c,dsp32cxx.out'.

7) To use an external sample clock, follow the basic guidelines below:

- Set any jumpers that may be required on the board or module itself. See the jumper list and/or hardware manufacturer documentation.
- Select External Sample Clock in the Hardware Manager dialog box or in DSPower function calls, or enter '-EXT' after the board designator in the Hypersignal System Configuration menu. This causes the high-level software to set the value of the shared variable "FS_MODE" as appropriate for external clock operation. If due to the design of the board no setting is possible, then the value of FS_MODE will be zero. If the high-level software does not set the value correctly, or the hardware has customized external sampling circuitry, then the FS_MODE variable should still be used to control the sampling rate. In this case, the DSP source code should be modified as needed.
- The actual sampling rate value is contained in the shared variable "FS_VALUE". The value is in Hz (Hertz); note that for C54xx boards, which use 16-bit wide memory accesses, the maximum value that can be represented is limited to 65535. This value is initialized by the high-level software, based on what is entered in the current menu (SAMPLING FREQ field).
- Enter sampling rate values in DSPower demo or application programs, or SAMPLING FREQ field values in Hypersignal software menus, to match the actual external sample clock rate. This will ensure that time and frequency unit readouts in all real-time and post-acquisition displays are accurate.

5.2 C67xx Source Code Interface

5.2.1 C67xx Example Applications

Some typical examples of applying the C67xx Source Code Interface software are listed below. In each case, the application is mentioned, followed by a suggested general approach to make source code modifications.

- Application: performing custom algorithm or feature extraction

  Method: add as application-specific C code (see section 5.1.2 below)
- Application: interfacing to custom or application-specific analog I/O hardware

  Method: modify the entry-point code in the appropriate ISR (interrupt service routine)

source code file; modify the analog I/O initialization code (note that this implies the remainder of the ISR can be left "as-is") in t67beg.asm or a add a file with a routine callable from t67beg.asm

- Application: stream real-time results to hard disk as a waveform file for post-analysis, either with or without application-specific C code processing of the real-time data

  Method: use appropriate setup in DSPower or Hypersignal software; see section 5.1.4 below. Hypersignal software or DSPower-Block Diagram software can be used to display, analyze, and measure the stored .tim or .wav file in both time and frequency domain

- Application: enable a "field debug and measurement" mode in a stand-alone DSP product

  Method: implement a flag in the DSP code to determine whether a host interface is present; if so, then debug and data collection functions can be enabled that are normally not

- Application: adding code to support interboard communication

  Method: modify existing ISR; add user-defined C/asm code to user-defined ISR

- Application: adding code to allow digital I/O to application-specific peripheral devices

  Method: modify existing ISR; add user-defined C/asm code to user-defined ISR

- Application: measuring algorithms or parts of algorithms against a particular real-time sampling rate

  Method: add buffer overrun detection to UserProc routine; see section 5.1.3.1

- Application: measuring MIPS requirements in a real-time system

  Method: check ISR buffer pointer value at end of calculation to determine remaining percentage of time left in current input frame; add to end of UserProc routine

- Application: evaluating the effects of finite word-length

  Method: add user-defined function to DSP/math function list in either C or asm code

5.2.2 Adding Application-Specific C Code

The following section focuses on adding application-specific C code as one or more C-callable functions which are called on a "buffer basis"; i.e. are called each time a new buffer of analog I/O data must be processed. This approach integrates the required C code into the existing C67xx framework at specific entry-points. For a more general approach, see section 5.2 below, Interface from Application C Code.

The process of adding application-specific C code is designed to isolate the algorithm represented by the C code from background and housekeeping tasks required to maintain host interface and real-time operation. For example, hardware and code initialization and setup, handling sample interrupts and interfacing to analog I/O hardware, communicating with the host,

and managing real-time data buffers are performed via pre-defined host and DSP software modules. These modules have well documented properties and methods accessible from host software and, if necessary, from C67xx initialization code. They can be accessed during setup and initialization and on an ongoing basis to allow dynamic control and configuration.

Maintaining reliable, high-speed data transfer host-interface capability in the face of code changes and analog I/O based real-time operation is a critical feature of the SigC67xx Development System; many software functions depend on it. Host-interface capabilities provided by DSPower programs and Hypersignal software include:

- controlling the DSP board
- board interrogation functions
- downloading executable code (COFF files)
- initializing “DSP properties” in the C67xx code
- transferring blocks of real-data between host and DSP board memory (either while code is stopped or running)

For more information on DSPower and Hypersignal capabilities, please see section 5.2.4, C67xx Source Code Access and Control from Host Programs.

5.2.3 Inserting Application-Specific C Code into the Real-Time Stream

Following is general information about how to add user-defined 'C' routines to real-time DSP code running under Hypersignal-Macro or Hypersignal-Acoustic software, or under C/C++, MATLAB, Visual Basic, or LabVIEW programs using the DSPower software API. Actual DSP 'C' code examples are given and differences between DSP chip families are accounted for with *#ifdef..#endif* constructs in the C code.

There are four (4) important ideas underlying the method for adding user-defined 'C' routines to real-time DSP code that is described in this section. These are:

- **hide details of the DSP chip and DSP board**--especially the analog I/O, which tends to vary extensively from board to board and can be extremely complex from a software point of view
- allow user-defined 'C' code to **process analog input/output data on a buffer, or "frame" basis**, and not deal with individual samples (but still allow a single-sample “framesize” in the case of control systems which require minimum system I/O delay
- allow **real-time recording of intermediate results to waveform file** or memory for debugging purposes
- **make the modification cycle, or "turn-around time", as fast as possible**

To achieve these goals, a routine called "UserProc" (in file userproc.c) has been defined in the standard C67xx Source Code which represents a general entry-point for user-defined real-time 'C' code. UserProc is called by the underlying C67xx Source Code framework after each analog input/output data buffer is filled and processed by the analog I/O drivers. This occurs at the interrupt service routine (ISR) level; a flag is set which indicates to foreground (non-interrupt) code that a new buffer is available. The “buffer ready” notification is handled transparently by the C67xx Source Code framework, which forwards the notification from the ISR to foreground C67xx code. Typically, a MODULn .asm file or an application-specific void main() routine running in the foreground receives the notification. In the case of a MODULn function, the notification can be (optionally) forwarded to host-side code, which receives the notification in the form of a callback message or as a release from a blocking WaitForBuffer() function call. Note that the notification occurs *before* analog input buffers are sent to host PC software, such as Hypersignal, or Windows programs using DSPower software (e.g. application-specific Visual C/C++, Visual Basic, MATLAB, or LabVIEW programs). This “insertion point” for application-specific C code allows both analog input buffers (before they reach host PC applications) and output buffers (after they are sent by host PC applications) to be modified by real-time DSP-based C and/or asm code.

5.2.3.1 UserProc: Real-Time 'C' Code Entry Point

Userproc.c represents a general entry-point for user-defined 'C' routines. It is called after each analog input/output data buffer is filled and processed by the analog I/O drivers and has been presented to the foreground processing layer in the C67xx framework. This type of operation is known as "frame-based real-time processing": background interrupt service routines assemble analog input samples into a series of frames, or buffers, which are then made available to foreground algorithms for processing on a "frame" basis. Note that the "current analog I/O drivers" are drivers that have been selected in the host PC software user-interface for the DSP/analog hardware that is currently installed in the system. In Hypersignal software, this is done using the System Config menu selection; in DSPower programs, this is done using the Hardware Manager dialog box or in DSAssignBoard function calls in the programs.

Parameters passed to userproc.c include a pointer to the just-acquired input buffer, a pointer to the *next* output buffer, length of each buffer in elements, and number of traces (channels) contained in each element. Note that each "element" could actually be more than one sample; for example a three-channel acquisition would cause buffers to contain interleaved data in the form ch0, ch1, ch2, ch0, ch1, ch2, etc.

Below is the default source code listing for the basic userproc.c routine; the default routine is configured for simple loopback operation (loop input back to output) before user-defined modification.

Default UserProc Routine

```
void UserProc(void* ptrIn, void* ptrOut, long nLen, short int nNumTrace) {

#ifdef defined(TMS320C3x) || defined(TMS320C4x) || defined(DSP56xxx) || defined(ADSP2106x)
  #define x ((long*)ptrIn)
```

```
  #define y ((long*)ptrOut)
#endif

#ifdef defined(DSP32C) || defined(DSP3210) || defined(TMS320C2x) || defined(TMS320C5x) || defined
(TMS320C67xx) || defined(ADSP21xx)
 #define x ((short int*)ptrIn)
 #define y ((short int*)ptrOut)
#endif

short int  n;

    for (n=0; n<nLen; n++) {

        y[n] = x[n];  /* loop input back to output */

#ifdef defined(TMS320C3x) || defined(TMS320C6x)
        y[n] <<= 16;
#endif
    }
}
```

Here are some important notes about the above code structure:

1) Each input buffer ("x") contains 16-bit A/D values represented as right-justified, signed, N-bit integer values, where N is the natural integer word-length of the DSP device.

2) Each output buffer ("y") contains 16-bit D/A values represented as right-justified, signed, N-bit integer values, where N is the natural integer word-length of the DSP device.

3) The nNumTrace parameter usually corresponds to the number of channels. However, it is more accurate to say "traces" rather than "channels" because of the logical-to-physical channel mapping allowed by the Channel List property specified in host-side software (Hypersignal or DSPower software). The Channel List property allows an arbitrary ordering of channels; for example, channel 0 may not correspond to trace zero (0).

   If the nNumTrace parameter is > 1, then data is stored as interleaved values, in repeating groups of nNumTrace values. For example,

   tr0,tr1,tr2,tr0,tr1,tr2, ...

   for 3-trace acquisition or signal output.

4) The xxxLINKC.CMD (or xxxLINKC.CTL) linker command (or control) file should be used if run-time 'C' support is needed (see section 5.2.7 below, C67xx Source Code Modification Process). Otherwise, the xxxLINK.CMD or xxxLINK.CTL linker command/control file versions can be used (without C run-time support). See section 5.2.7 below, C67xx Source Code Modification Process, for details. Run-time 'C' support is made necessary by adding functions or routines to the C code which require math and other support libraries. Typically, the library file "rtsPP.lib" would be required, where PP is a name assigned by the DSP chip manufacturer (for example “67” for C67xx, "30" for C3x, "56" for 56xxx, etc.). The xxxLINKC.CMD and xxxLINKC.CTL command/control files link in the appropriate rtsPP.lib file.

5) ***Important Operational Note:*** UserProc must complete its processing *before* the next input/output buffer pair is ready! Otherwise, real-time operation does not occur.

5.2.3.2 UserProc Application Examples

The UserProc entry-point represents a method to integrate application-specific C code into the existing C67xx real-time framework at specific points, or "hooks". The C code takes the form of "callback" functions which are called by the C67xx framework on a "buffer basis"; i.e. are called each time a new buffer of analog I/O data must be processed. For a more general approach, see section 5.3 below, Interface from Application C Code. A few examples of UserProc functions are given below.

Signal Arithmetic Example

A very simple example of user-defined real-time 'C' code is basic signal arithmetic requiring no recursive input (i.e. no input from previous output samples). In this example, the DSP algorithm scales and adds a constant to input channel 0 data, and also loops the data back to the output. The difference equation for this example could look something like:

```
y[n] = 0.75*x[n] + 1000
```

First, userproc.c is modified to look like:

```
void UserProc(void* ptrIn, void* ptrOut, long nLen, short int nNumTrace) {

#ifdef defined(TMS320C3x) || defined(TMS320C4x) || defined(DSP56xxx) || defined(ADSP2106x)
  #define x ((long*)ptrIn)
  #define y ((long*)ptrOut)
#endif

#ifdef defined(DSP32C) || defined(DSP3210) || defined(TMS320C2x) || defined(TMS320C6x) || defined
(TMS320C67xx) || defined(ADSP21xx)
 #define x ((short int*)ptrIn)
 #define y ((short int*)ptrOut)
#endif

short int  n;

    for (n=0; n<nLen; n++) {

        x[n] = 0.75*x[n] + 1000;
        y[n] = x[n];

#ifdef defined(TMS320C3x) || defined(TMS320C4x)
        y[n] <<= 16;
#endif
    }
}
```

Second, the following steps are performed:

- Compile userproc.c. As an example of how to do this, the batch file "c.bat" is included which invokes the manufacturer's 'C' compiler from a DOS command prompt.

- Run the **lnk.bat** file if run-time support is *not* needed; i.e. if the rtsPP.lib file is not needed (see section 5.2.7 below, C67xx Source Code Modification Process). Otherwise, run the **lnkc.bat** file.
- Copy the resulting .out file to the Hypersignal directory. The default .out filename is as specified in the Hypersignal-Macro Series Hardware Reference Guide; other filenames can be specified by providing the lnk.bat or lnkc.bat files a filename parameter, for example:

```
    lnkc test.out
```
or
```
    lnk test
```

  which will both produce an output file of "test.out".

- 4) Run host PC software to activate and test newly modified DSP code, using Hypersignal software or DSPower program (i.e. C/C++, MATLAB, Visual Basic or LabVIEW program that calls DSPower functions).
  - **To activate UserProc in Hypersignal operation, enter:**

```
        'a,rt,r'
```
or
```
        'a,rtd,r'
```

    in the ACQ/GEN field in the Analog Conversion function. The 'rt' entry causes real-time simultaneous input/output operation (mode 6; see section 5.2.5 below, C67xx Source Code Interface Structure). The 'rtd' entry works the same way, except recording to waveform file is enabled. Maximum sampling rates at which continuous operation (no gaps or data discontinuity) can be achieved are lower when file recording is enabled. In all cases, maximum sampling rates at which real-time operation can be maintained are heavily system dependent. Important factors include the DSP chip speed, DSP onboard SRAM access-time and efficiency of user-defined 'C' code. In the case of waveform file recording, factors also include host PC speed, disk drive speed and controller type, ISA or PCI bus speed, etc. For a detailed discussion of the factors influencing maximum continuous sampling rate to file, please see the "Super Guide" online help file which ships with DSPower software products.

    Important Note: if the name of the DSP program file has been changed from the default (by renaming the output of lnkc.bat, or by giving lnkc.bat a filename parameter), Hypersignal must know about the change before running Analog Conversion. To do this, suffix the filename to the ANALOG CONVERSION field in the System Config menu. For example, this could result in entries like 'IIC31-B-50,test.out', 'TBS56-B,AC3.OUT', or 'sig32c,dsp32cxx.out'.

  - **To activate UserProc when using DSPower software, there are two methods:**
    - **DSAcquireWvfrmFile Approach.** Use either "rt" or "rtd" as the stimulus filename szStimName structure member) in the CONVERSIONINFO structure passed to the DSAcquireWvfrmFile function. This instructs Hypersignal to replace stimulus filename output with real-time processed

data. The "rt" value causes indefinite real-time operation (until the engine is idled; see the dtape.cpp source code file for an example of how to use the DSSendEngineCommand(hEngine, DS_SEC_IDLE, DS_EEF_SYNC) function call). The "rtd" value causes real-time operation with continuous recording to waveform file of the analog input data until the NumSamples structure member is reached. The record-to-disk option is provided as a means to post-analyze, in detail, data that has been processed in real-time by userproc.c. Waveform file data can be analyzed using a Hypersignal or DSPower-Block Diagram waveform display function.

Also, the stimulus mode (szStimMode stucture member) value should be set to "R", which instructs the DSP code to perform repetitive frame output until the waveform file is complete or the operation is aborted.

- **DSP Code Control Approach.** Modify the rtcode.cpp file as needed. The rtcode.cpp file shows an example of initializing DSP code for UserProc operation with DSSetDSPProperty calls. The key steps are to choose operating mode 6 (set DSP_OPMODE property to 6) and initialize the value of the DSP_FRMSIZ property to the length of each analog input buffer. Normally the DSP_FRMSIZ value should be the same as the DSP_BUFLEN property. Note that with this approach, continuous recording of input or output buffer data to waveform file is not active; to make this functionality active, it would have to be added to the rtcode.cpp file.

Running Sum Example

A slightly more complex example is to combine basic signal arithmetic with recursive input (i.e. input from previous output samples). In this example, the DSP algorithm performs a running sum on input channel 0 data, with no loopback to the output. The difference equation for this example could look something like:

```
y[n] = y[n-1] + (x[n] - x[n-M])/M
```

where M is the length of the running sum, in samples. First, userproc.c is modified to look like:

```
#define N 4096        /* define N as maximum buffer size */
#define M 32          /* define M as length of running sum */

short int ySum[N];    /* create storage of size of one buffer, assume it
                         is initialized to zero */

short int xPrev[M];   /* create storage of length of running sum, to hold
                         samples at end of each previous buffer ("memory"
                         between successive frames) */

void UserProc(void* ptrIn, void* ptrOut, long nLen, short int nNumTrace) {

#ifdef defined(TMS320C3x) || defined(TMS320C4x) || defined(DSP56xxx) || defined(ADSP2106x)
  #define x ((long*)ptrIn)
  #define y ((long*)ptrOut)
#endif
```

```
#ifdef defined(DSP32C) || defined(DSP3210) || defined(TMS320C2x) || defined(TMS320C6x) || defined
(TMS320C67xx) || defined(ADSP21xx)
 #define x ((short int*)ptrIn)
 #define y ((short int*)ptrOut)
#endif

short int     n,xn;
long static   lSum = 0;

    for (n=0; n<nLen; n++) {

        if (n - M < 0) xn = xPrev[n];    /* use "memory" if index is before
                                            start of current buffer */
        else xn = x[n-M];

        lSum += x[n] - xn;               /* calculate x term sum */

        ySum[n] += (short int)(lSum/M);  /* calculate running sum */
    }

    for (n=0; n<M; n++) xPrev[n] = x[nLen-n];  /* store last M points
                                                  for use by next frame */

    for (n=0; n<nLen; n++) x[n] = ySum[n];     /* store processed data
                                                  in input buffer */
}
```

After modifying userproc.c, see the steps outlined in section 5.2.7 below, C67xx Source Code Modification Process.

Notes

1) The #define constructs above for non-C67xx DSP types can be removed.

2) The above use of a long division intermediate calculation above (lSum/M) is not suggested for real-time processing; it is used only to make the example general and portable across different DSP devices. A better approach would be to make M a power of 2, so the division could be achieved by shifting the lSum term right.

5.2.4 C67xx Source Code Access from Host Programs

There are three (3) general methods of accessing C67xx source code from host programs:

1) Hypersignal-Macro (and Hypersignal-Macro Ex and Hypersignal-Acoustic) software
2) DSPower software
3) DSPower-Block Diagram and Real-Time Composer software

The first two (2) methods above allow flexible user-specified access and control of the DSP(s) at all levels, including debug and measurement functions; these are covered in the sections below. The third method automates DSP hardware control based on graphical signal flow diagrams; the Real-Time Composer software is currently in beta release, and so is not covered in this document at this time.

5.2.4.1 C67xx Source Code Access and Control from Hypersignal Software

The Hypersignal-Macro series software packages contain capabilities that facilitate access and control of Hypersignal DSP source code containing user-defined modifications or additions, including:

1) The ability to specify different executable DSP files to load and run when a DSP-based function or display is invoked from Hypersignal. In the System Configuration menu, entering a filename, such as "new.out", after the board designator will override the default DSP filename that Hypersignal uses for default operation.
2) Macro language functions including InitBoard, LoadBoard, RunBoard, ResetBoard, WriteDSPVal and WriteDSPMem that allow macro programs to access DSP/data acquisition hardware as needed.
3) Instrumentation functions which allow record-to-file and display of real-time data from the DSP/data acquisition hardware. For example, a mode can be selected in the Analog Conversion function that allows buffers, either raw or processed, to be uploaded from the shared memory on the DSP board and stored continuously to disk file. For detailed information on this procedure, see section 5.2.3.2, UserProc Application Examples.

In general, maintaining the host interface capability in the C67xx source code allows Hypersignal functions to continue to be enabled and usable. This allows Hypersignal to store data in .tim and .wav waveform file formats, and display and analyze stored data. Time domain and frequency domain displays, DSP operations such as FFT, filtering, difference equations, etc. continue to be accessible with results produced by the C67xx code. See section 5.2.5.2 above, C67xx Source Code Interface Host PC Communication, for more information.

For more information about the functions listed in item 2) above, see Hypersignal-Macro Series Users Manual section 7.10, Macro Commands and Functions.

Notes

1) Both the WriteDSPMem and WriteDSPVal functions always make full-word accesses, depending on the natural word-length of the DSP device. This can affect the address value used, if the DSP is capable of accessing data pieces smaller than full-word length.

2) Both WriteDSPMem and WriteDSPVal functions are capable of accessing all memory types supported by the DSP chip. For a complete list of internal Hypersignal values that can be written to arbitrary DSP source code memory, see Signalogic DSP Software Reference Guide section 3.3, Hypersignal DSP Source Code Variable List. For fully detailed information about the syntax and usage of these functions, see Hypersignal-Macro Series Users Manual section 7.10, Macro Commands and Functions.

5.2.4.2 C67xx Source Code Access and Control from DirectDSP Programs

DSPower software is an interface layer between host application programs and DSP/data acquisition hardware and DSP algorithm code. On the application side, it can be incorporated as DLLs and ActiveX controls for programs written in Visual C/C++ and Borland C/C++, MATLAB, Visual Basic, and LabVIEW. On the DSP side, it communicates with drivers for more than 100+ types of DSP/data acquisition hardware; also, it interfaces to Hypersignal-Macro software, MATLAB, and other user-defined drivers and "DSP engines".

DSPower software has two (2) main purposes:

- provide a direct, flexible, but still uniform interface to DSP/data acquisition hardware from popular host environments
- provide a well documented method to interface to DSP algorithms executing in real-time

It is the latter purpose that is the focus of this section. When communicating with DSP code from one of the supported host environments, DSPower software provides the following low-level DSP code-oriented functions:

- download DSP program files (COFF files) produced by the DSP vendor's compiler, assembler, and linker tools
- set and get DSP properties, both pre-defined and user-defined
- transfer data buffers between host memory and onchip and offchip memory on the DSP board, including various data formats, automatic format conversion options, and the ability to transfer data while DSP code continues to execute in real-time
- use COFF symbol information to determine physical addresses of variables, buffers, and vectors in the DSP code
- provide debug capability, including breakpoint, single-step, processor status display, and data display in text or graphical waveform formats

In addition, DSPower software provides the following high-level DSP code oriented functions:

- real-time buffer management, including "buffer ready" messages and callback message processing
- pre-defined instrumentation functions such as digital scope, spectrum analyzer, streaming to disk
- full-featured analog I/O support when applicable

Note that the debug capability mentioned above is not hardware based as with standard JTAG debuggers. Rather, this capability depends on software "trap" interrupts to handle breakpoints

and single-step, so it is not “true software-independent debug”. However, unlike JTAG, the DSPower debug capability can transfer large amounts of data without stopping the processor(s).

There are two main approaches to running DSP code from DSPower programs:

- “talker only”
- use pre-defined default Signalogic DSP framework as all or some part of the algorithm code

The “talker only” approach refers to the most general case: only a small interrupt driven host communication kernel (a talker; see section 5.2.5.2, C67xx Source Code Interface Host PC Communication) is used and all other DSP code is application-specific. In this case, no assumptions can be made by DSPower programs about pre-defined properties, buffer locations, code initialization, analog I/O drivers and pre-defined operating modes. This allows DSPower core functionality (COFF download, data transfer, board control and interrogation, etc.) to remain intact, but disables higher level DSPower functions such as buffer management, instrumentation functions such as digital scope, spectrum analyzer, streaming to disk, etc.

Note that the latter case, where the Signalogic C67xx framework is utilized, requires the talker to be enabled, and depends on it for host communication.

The sections below provide a summary of the DSPower API, and also show code examples for several common DSP board functions, including for COFF file download, DSP property initialization, and several types of data transfer. Source code examples are provided for host environments C/C++, MATLAB and Visual Basic.

5.2.4.2.1 DirectDSP API Summary

Below is a partial summary of function calls in the DSPower software package:

Low-Level Functions

Board and processor control, board register and peripheral access, block memory transfers, COFF file download, IEEE-to-DSP conversion

```
DSInitBoard
DSResetBoard, DSResetProcessor
DSRunBoard, DSRunProcessor
DSHoldBoard, DSHoldProcessor
DSDisableBoard
```

DSLoadFileProcessor, DSLoadFileBoard

```
DSPutMem
DSGetMem
DSSetDSPProperty
DSGetDSPProperty

DSWriteBoardReg
```

Buffer and Semaphore Management

buffer-ready message callback window registration, wait for buffer

```
DSRegisterEngineMsgWnd
DSWaitForBuffer
DSWaitForFlag
DSCancelWait

DSGetBufferInfo
DSGetFlagInfo
```

Callback Messages

buffer- and flag-ready, status/error, and parameter callback messages

```
WM_DSPENGINE_BUFRDY
```

DSReadBoardReg

DSIEEEToDSP
DSDSPToIEEE

Autocalculation and Autodetection

autodetect board memory size and architecture, calculate neares allowable sampling frequency

DSGetBoardClass
DSCalcSampFreq
DSGetMemSize
DSGetMemArch
DSGetWordLength

Hardware Management

hardware driver list management, pop-up hardware manager dialog box

DSShowHardwareSelectorDialog

DSGetHWMgrNumEntries
DSGetHWMgrEntryIndex

DSGetHwMgrEntry, DSSetHWMgrEntry

High-Level Functions

record .TIM waveform files, playback .TIM and .WAV waveform files, activate real-time filtering, stimulus & response measurement, continuous signal generation, exec arbitrary DSP functions

DSAcquireWvfrmFile(CONVERSIONINFO FAR*)
DSGenerateWvfrmFile(CONVERSIONINFO FAR*)

CONVERSIONINFO Structure

waveform filename
number of channels, channel list
sampling rate
number of samples
output start sample index
gain / attenuation list
digital offset and digital scale
output repeat
input loopback
real-time filter 1 filename
real-time filter 2 filename
trigger mode, level, and delay
number of trigger channels,
trigger channel list
stimulus filename, mode, and delay

WM_DSPENGINE_FLAGRDY

WM_DSPENGINE_ENGINESTATE
WM_DSPENGINE_FUNCTIONERR
WM_DSPENGINE_CODEGENERROR
WM_DSPENGINE_COMPILEERROR

WM_DSPENGINE_RUNTIMEERROR

WM_DSPENGINE_BUFSIZE
WM_DSPENGINE_SAMPFREQ
WM_DSPENGINE_BUFNUM
WM_DSPENGINE_AMPLEVEL

Engine Management

Driver and engine management, error/status window & reporting

DSEngineOpen
DSEngineClose
DSGetEngineErrorStatus
DSGetEngineUsage
DSEngineSendCommand
DSSetEngineWaveformPath
DSSetEngineTemplatePath

DSShowEngMgrStatusWindow
DSAddEngMgrStatusLine
DSHideEngMgrStatusWindow

Board and Processor Management

Board and processor handle allocation, get board info, get/set board I/O and mem. base addresses, get/set bus type

DSAssignBoard
DSFreeBoard

DSGetBoardInfo

DSAssignProcessor
DSFreeProcessor

DSGetBoardBaseAddr, DSSetBoardBaseAddr
DSGetBoardBusType, DSSetBoardBusType

Memory Function Constants

DSGetMemArch,DSPutMem, DSGetMem constants
DS_GMA_LINEAR
DS_GMA_HARVARD
DS_GMA_VECTOR

DS_GM_VECTOR_DATA_X
DS_GM_VECTOR_DATA_Y
DS_GM_LINEAR_PROGRAM
DS_GM_LINEAR_DATA

DS_GM_SIZE8
DS_GM_SIZE16
DS_GM_SIZE24
DS_GM_SIZE32

5.2.4.2.2 C/C++ Examples in DirectDSP

The C/C++ code excerpts below show examples of:

- ✓ Hardware Manager dialog box display
- ✓ Driver and board initialization
- ✓ DSP program file download
- ✓ DSP property initialization
- ✓ Waveform generation buffer calculation and download
- ✓ Run board; run one or more processors
- ✓ Buffer-ready callback message and I/O data buffer processing
- ✓ Cleanup (board shutdown)

For complete C/C++ source code examples, see the following examples included with DSPower software:

Source Code Example Program Type	**Filename**
Debugger example	sdebug.cpp
Digital oscilloscope	dscope.cpp
Digital tape recorder	dtape.cpp
Real-time C code scope	rtcode.cpp
Simultaneous real-time analog I/O	sim_io.cpp
Spectrum analyzer	spcana.cpp
Strip-chart recorder	strip.cpp

Hardware Manager Dialog Box

```
// show hardware manager dialog (Hardware Manager); returns board
// designator string

// note:  if hardware choice is fixed, then skip this call and use
// correct board designator string in DSAssignBoard call below

   if (DSShowHardwareSelectorDlg(NULL, szBoard) == IDCANCEL)
      goto cleanup;
```

Driver and Board Initialization

The driver and board initialization functions below do not include error checking. For complete examples with error checking, please consult dscope.cpp, sim_io.cpp, and other C/C++ example source code files included with DSPower software.

```
// open engine or driver

   hEngine = DSEngineOpen(DS_EO_HSM, NULL, NULL);  // first try to open
```

```
                                                    // Hypersignal-Macro or
                                                    // Hypersignal-Macro EX as
                                                    // engine

   if (!hEngine) {

       hEngine = DSEngineOpen(DS_EO_HSA, NULL, NULL);  // if that doesn't
                                                       // work, try
                                                       // Hypersignal-Acoustic
    }

// note:  using a string name instead of the pre-defined “DS_EO_xxx” constants
// above will access a user-defined driver.  Drivers must export certain
// baseline functions required by DSPower software.  Please ask Signalogic
// for the “OEM DSPower Software Development Guide” document.

// assign a board handle:  engine handle, board designator, bus type, IO base
// addr, Mem base addr

   hBoard = DSAssignBoard(hEngine, szBoard, NULL, NULL, NULL);


// initialize the board; make sure it's installed, reset all processors

   fBoardInitialized = DSInitBoard(hBoard);
```

DSP Program File Download

```
// load executable DSP code file (typically a COFF file produced by DSP
// manufacturer's linker)

   DSLoadFileProcessor(hBoard, NULL, 0x01);  // load default file for the
                                             // board type (processor 0 only)

// interrogate engine for board type values

   wBoardClass = DSGetBoardClass(hBoard);


// determine sampling rate ctrl. reg. value, and actual rate (closest rate
// possible to desired); CalcSampFreq returns ctrl. reg. value directly, uses
// ptr to return actual sampling frequency (in Hz); demo assumes 1 channel

   dwFsMode = DSCalcSampFreq(hBoard, FsDesired, 1, &ChanList, &FsActual);

// get memory architecture

   uMemArch = DSGetMemArch(hBoard);


   if (uMemArch == DS_GMA_VECTOR)
      BufMul = 1;
   else
      BufMul = 2;

// get address of input time domain data

   dwBufferBaseAddr = DSGetDSPPropertyMem(hBoard, DSP_TIMDATAADDR);

// get address of output time domain data

dwOutBufferBaseAddr = dwBufferBaseAddr + BufMul*wBuflen;

// get the memory size, (note that this currently has to be done after LoadFile)

   dwMemSize = DSGetMemSize(hBoard, 0x01);  // processor 0 only
```

```
// reset the DSP board (should already be in reset state; processor 0 only)

   DSResetProcessor(hBoard, 0x01);
```

DSP Property Initialization

The DSP property initialization examples below may not be complete for all pre-defined operating modes in the DSP Source Code Interface. Please also consult the Visual Basic and MATLAB examples for possible additional property examples.

```
// initialize DSP properties required for real-time, continuous simultaneous
// analog I/O operation

   DSSetDSPProperty(hBoard, DSP_BOARDCLASS, wBoardClass & 0x0ff);

   DSSetDSPProperty(hBoard, DSP_BOARDSUBCLASS, wBoardClass >> 8);

   DSSetDSPProperty(hBoard, DSP_OPMODE, 6);     // simultaneous I/O is mode 6

   DSSetDSPProperty(hBoard, DSP_FILTTYPE1, 0);  // trace 1 real-time filter off

   DSSetDSPProperty(hBoard, DSP_FILTTYPE2, 0);  // trace 2 real-time filter off

   DSSetDSPProperty(hBoard, DSP_TRIGLEVEL, 0);       // free-run triggering
   DSSetDSPProperty(hBoard, DSP_TRIGCHANLIST, 0);

   DSSetDSPProperty(hBoard, DSP_BUFLEN, wBuflen);    // input buffer size
   DSSetDSPProperty(hBoard, DSP_STMBUFLEN, wBuflen); // output buffer size

    DSSetDSPProperty(hBoard, DSP_STMDATAADDR,
                     dwOutBufferBaseAddr);      // output buffer base address

   DSSetDSPProperty(hBoard, DSP_HOSTBUFNUM, 0); // host buffer number
   DSSetDSPProperty(hBoard, DSP_BUFNUM, 0);     // DSP buffer number

   DSSetDSPProperty(hBoard, DSP_CHANLIST, 0);   // channel list( each channel is
                                                // 4bits, start from left; first
                                                // channel is 0)
   DSSetDSPProperty(hBoard, DSP_NUMCHAN, 1);    // number of channels

   DSSetDSPProperty(hBoard, DSP_GAINLIST, 0);   // gain list

   DSSetDSPProperty(hBoard, DSP_FSMODE, dwFsMode);  // sampling rate control
                                                    // register (mode value)

   DSSetDSPProperty(hBoard, DSP_FSVALUE, (long)FsActual));  // actual sampling
                                                            // rate (in Hz)


// set output digital scale (multiply) factor and offset(add) to DSP code

   if ( DSGetBoardInfo(hBoard, DS_GBI_DSPWORDLENGTH) == 24) {
      nShift = 7;
      nShift2 = 8;

   }
   else {
      nShift = 0;
      nShift2 = 0;
   }


// set analog input/output scale factors (scale factors are implemented
// in Q-8 in DSP code)

   DSSetDSPProperty(hBoard, DSP_SCALEIN, 256*InputScaleFactor*(1 << nShift));
```

```
   DSSetDSPProperty(hBoard, DSP_SCALEOUT, 256*OutputScaleFactor*(1 << nShift));

// set analog input/output offsets

   DSSetDSPProperty(hBoard, DSP_OFFSETIN, InputOffset*(1 << nShift2));
   DSSetDSPProperty(hBoard, DSP_OFFSETOUT, OutputOffset*(1 << nShift2));
```

Waveform Generation Buffer Calculation and Download

```
// calculate buffer size as approximately holding integral number
// of periods of output sine wave, based on actual sampling frequency,
// to form continuous boundary between output buffers (see output waveform
// calculation and download/storage to board below)

// note the value of buffer size also depends on the type of board;
// different boards have different maximum buffer size depending on
// the amount of onchip/offchip memory available; the value we are
// using is typically small in order to allow this demo program to
// work on a wide range of boards; for better streaming, continuous
// data transfer performance, the buffer size should be increased,
// if the board will allow it; for example, the calculation below
// could be made = / 1, to increase both accuracy and performance

   wBuflen = (int)(FsActual / 10);

   buffer1 = (short*)GlobalLock(hBuf1);
   buffer2 = (short*)GlobalLock(hBuf2);

// Create data buffer; note that data must be scaled to be reasonable values
// for D/A converter(s)

   for (n=0; n < wBuflen; n++){
      buffer1[n] = (short)(10000*sin(2*pi*n*1000/FsActual));
      buffer2[n] = (short)(10000*sin(2*pi*n*1000/FsActual));
   }

// pre_load DSP board memory with data buffers;
// both buffers must perloaded for continuous operation.
// note that download depends on whether board has linear or Harvard memory
// architecture

   if (uMemArch == DS_GMA_VECTOR){ // board has vector memory architecture

      uStatus = DSPutMem(hBoard, DS_GM_VECTOR_DATA_X, dwOutBufferBaseAddr,
                         DS_GM_SIZE16, buffer1, wBuflen);
      uStatus = DSPutMem(hBoard, DS_GM_VECTOR_DATA_Y, dwOutBufferBaseAddr,
                         DS_GM_SIZE16, buffer2, wBuflen);
   }
   else{  // board has linear memory architecture

      uStatus = DSPutMem(hBoard, DS_GM_LINEAR_DATA_RT, dwOutBufferBaseAddr,
                         DS_GM_SIZE16, buffer1, wBuflen);
      uStatus = DSPutMem(hBoard, DS_GM_LINEAR_DATA_RT,
                         dwOutBufferBaseAddr+wBuflen, DS_GM_SIZE16,
                         buffer2, wBuflen);
   }

   GlobalUnlock(hBuf1);
   GlobalUnlock(hBuf2);

   GlobalFree(hBuf1);
   GlobalFree(hBuf2);
```

Run Board; Run One or More Processors

```
// run board/processor examples

   if (fMultiProcessor)
```

```
      DSRunProcessor(hBoard, dwProcList);
   else if (fMultiProcessor && fFirstProcActiveOnly)
     DSRunProcessor(hBoard, 0x01);
   else if (fMultiProcessor && fAllProcActive)
     DSRunBoard(hBoard);
   else if (fSingleProcessor)
    DSRunBoard(hBoard);
```

Buffer-Ready Callback Message and I/O Data Buffer Processing

```
case WM_DSPENGINE_BUFRDY:  // message sent by DSP engine or driver when data
                           // buffer is ready

   pBuffer = (short int*)GlobalLock(hBuf);


   if (uMemArch == DS_GMA_VECTOR) {  // vector data memory

      if (nCurBuf == 0)
         uStatus = DSGetMem(hBoard, DS_GM_VECTOR_DATA_X, dwBufferBaseAddr,
                            DS_GM_SIZE16, pBuffer, wBuflen);
      else
         uStatus = DSGetMem(hBoard, DS_GM_VECTOR_DATA_Y, dwBufferBaseAddr,
                            DS_GM_SIZE16, pBuffer, wBuflen);
      }
      else  // linear data/prog memory, or modified harvard arch. with linear
            // data memory
         uStatus = DSGetMem(hBoard, DS_GM_LINEAR_DATA_RT,
                            dwBufferBaseAddr+nCurBuf*wBuflen, DS_GM_SIZE16,
                            pBuffer, wBuflen);

      if (!uStatus)
        DSMessageBox(hwnd,
                     "DSGetMem:  problem with point transfer", szApp, MB_OK);

/* Output the acquired data
   comment the PutMem section if the pre-loaded data (see WinMain above) should
   continue to be output repetitively; otherwise, this section will loop input
   buffers to output, with a one buffer delay
*/

   if (uMemArch == DS_GMA_VECTOR) {  // vector data memory

      if (nCurBuf == 0)
         uStatus = DSPutMem(hBoard, DS_GM_VECTOR_DATA_X, dwOutBufferBaseAddr,
         DS_GM_SIZE16, pBuffer, wBuflen);

      else
         uStatus = DSPutMem(hBoard, DS_GM_VECTOR_DATA_Y, dwOutBufferBaseAddr,
         DS_GM_SIZE16, pBuffer, wBuflen);

   }
   else { // linear data/prog memory, or modified harvard arch. with linear
            data memory

      uStatus = DSPutMem(hBoard, DS_GM_LINEAR_DATA_RT,
                         dwOutBufferBaseAddr+nCurBuf*wBuflen, DS_GM_SIZE16,
                         pBuffer, wBuflen);

   }

   GlobalUnlock(hBuf);

   if (!uStatus)
     DSMessageBox(hwnd,
                  "DSGetMem:  problem with point transfer", szApp, MB_OK);
   else FirstBufferRcvd = TRUE;

   nCurBuf ^= 1;  // switch buffers
```

```
// wait for next buffer; note this function does not actually wait unless the
// DS_WFB_SYNC flag is used; in the example below, it sets up a callback message

   DSWaitForBuffer(hBoard, nCurBuf, NULL, DS_WFB_POLLED);


// update the screen with new data

   MyPlotFunction(hBuf);
```

Cleanup (Board Shutdown)

```
    if (hBoard != NULL) {

      if (fBoardInitialized) DSDisableBoard(hBoard);  // disable board (all
                                                      // processors)
      DSFreeBoard(hBoard);
   }

   if (hEngine != NULL) DSEngineClose(hEngine);
```

5.2.4.2.3 MATLAB Examples in DirectDSP

The MATLAB code excerpts below show examples of:

- ✓ Hardware Manager dialog box display
- ✓ Driver and board initialization
- ✓ DSP program file download
- ✓ DSP property initialization
- ✓ Waveform generation buffer calculation and download
- ✓ Run board; run one or more processors
- ✓ Continuous I/O data buffer processing
- ✓ Cleanup (board shutdown)

For complete MATLAB source code examples, see the following .m file examples included with DSPower software:

Source Code Example Program Type	**Filename**
Digital oscilloscope	dscope.m
Data format conversion	hwtest3.m
Continuous buffer analog input	hwtest5.m
Continuous buffer analog output	hwtest4.m
Frame-based analog buffer input and plot	hwtest2.m
Simultaneous real-time analog I/O	hwtest6.m

Waveform file playback	hwtest1.m

Hardware Manager Dialog Box

```
% show hardware manager dialog box ("Hardware Manager"); returns board
% designator
% string selected by user
% note:  if hardware choice is fixed, then skip this call and use correct board
% designator
% string in AssignBoard call below

 [fSelect, BoardStr] = ShowHwSelector(NULL);

 if (fSelect == IDCANCEL)  % user canceled?
  return;
 end
```

Driver and Board Initialization

```
% open dsp engine or driver

% default: first try to open Hypersignal-Macro (or Macro Ex) as engine

hEngine = EngineOpen(DS_EO_HSM, NULL, NULL, DS_EO_NOTVISIBLE);

if (hEngine == NULL)

% if that doesn't work, try Hypersignal-Acoustic

    hEngine = EngineOpen(DS_EO_HSA, NULL, NULL, DS_EO_NOTVISIBLE);

    if (hEngine == NULL)

        errstr = ['EngineOpen failed; error code = ' num2str(GetHwLibErrStat)];
        disp(errstr);

        return;  % abort
    end
end

% note:  using a string name instead of the pre-defined “DS_EO_xxx” constants
% above will open a user-defined driver.  Drivers must export certain
% baseline functions required by DSPower software.  Please ask Signalogic
% for the “OEM DSPower Software Development Guide” document.


% assign a board handle:  engine handle, board designator string, instruction
% set, IO base addr, Mem base addr

hBoard = AssignBoard(hEngine, BoardStr, NULL, NULL, NULL);  % NULL == use
                                                            % default values


% initialize the board; make sure it's installed and responding

fBoardInitialized = InitBoard(hBoard);

if (fBoardInitialized == FALSE)

    errstr = ['InitBoard failed; error code = ' num2str(GetEngErrStat(hEngine))];
    disp(errstr);

    if (hBoard ~= NULL)
        FreeBoard(hBoard);
    end

    if (hEngine ~= NULL)
```

```
        EngineClose(hEngine);
    end

    return;  % abort
end
```

DSP Program File Download

```
% load executable DSP code file (usually a COFF file produced by DSP
% manufacturer's linker)

if (LoadProcessor(hBoard, NULL, 1) == NULL)  % load default file for the board
                                             % type (processor 0 only)

    errstr = ['LoadProcessor failed; error code = ' num2str(GetEngErrStat(hEngine))];
    disp(errstr);

    if (hBoard ~= NULL)
        FreeBoard(hBoard);
    end

    if (hEngine ~= NULL)
        EngineClose(hEngine);
    end

    return;  % abort
end
```

DSP Property Initialization

The DSP property initialization examples below may not be complete for all pre-defined operating modes in the DSP Source Code Interface. Please also consult the C/C++ and Visual Basic examples for possible additional property examples.

```
% prepare hardware and data transfer values, allocate memory to receive buffers

BufNum = 0;               % start with buffer 0
NumChan = 1;              % this example uses one channel
InputOffset  = 0.0;       % DSP input offset
OutputOffset = 0.0;       % DSP output offset
InputScaleFactor  = 1.0;  % DSP input scale factor
OutputScaleFactor = 1.0;  % DSP output scale factor
desiredSampFreq = 20000;  % desired sampling rate
desiredOutFreq = 1000;    % desired output frequency

BufSize = min(desiredSampFreq*5, 4096);  % default buffer size

BoardClass = GetClass(hBoard);       % board classification
MemArch = GetMemArch(hBoard);        % board memory architecture

if (MemArch == DS_GMA_VECTOR)
 BufMul = 1;                         % multiplier for BufSize
else
 BufMul = 2;
end

TimBaseAddr = GetVarMem(hBoard, DSP_TIMDATAADDR);  % base address of input
OutBaseAddr = TimBaseAddr+BufMul*BufSize;          % base address of output

[FsMode, FsActual] = CalcSampFreq(hBoard, desiredSampFreq, NumChan, NULL);

buf = zeros(1, BufSize);       % allocate buffer memory  ***
buf1 = zeros(1, BufSize);
buf2 = zeros(1, BufSize);
buf_plot = zeros(1, BufSize);

ResetBoard(hBoard);  % reset board (should already be in reset state)
```

```
% write property values to DSP code on board

SetDSPProperty(hBoard, DSP_BOARDCLASS, rem(BoardClass, 256));
SetDSPProperty(hBoard, DSP_BOARDSUBCLASS, fix(BoardClass/256));

SetDSPProperty(hBoard, DSP_OPMODE, 6);     % continuous input/output mode

SetDSPProperty(hBoard, DSP_FILTTYPE1, 0);  % disable trace 1 real-time filter
SetDSPProperty(hBoard, DSP_FILTTYPE2, 0);  % disable trace 2 real-time filter

SetDSPProperty(hBoard, DSP_BUFLEN, BufSize);    % input buffer size
SetDSPProperty(hBoard, DSP_STMBUFLEN, BufSize); % output buffer size

SetDSPProperty(hBoard, DSP_STMDATAADDR, OutBaseAddr);  % output buffer base
                                                       % address

SetDSPProperty(hBoard, DSP_HOSTBUFNUM, 0);      % host buffer number
SetDSPProperty(hBoard, DSP_BUFNUM, 0);          % DSP buffer number

SetDSPProperty(hBoard, DSP_CHANLIST, 0);        % channel list (each channel is 4 bits, start
from left; first channel is 0)
SetDSPProperty(hBoard, DSP_NUMCHAN, NumChan);   % number of channels

SetDSPProperty(hBoard, DSP_GAINLIST, 0);        % gain/attenuation list

SetDSPProperty(hBoard, DSP_TRIGLEVEL, 0);              % free-run triggering
SetDSPProperty(hBoard, DSP_TRIGCHANLIST, 0);         % triger channel

% set output digital scale (multiply) factor and offset (add) to DSP code

if (GetBoardInfo(hBoard, DS_GBI_DSPWORDLENGTH) == 24)
    nShift = 7;    % weird efficiency thing required by DSP56xxx boards
    nShift2 = 8;
else
    nShift = 0;
    nShift2 = 0;
end

% write analog I/O scale factors (scale factors are implemented as Q-8 in DSP
% code)

SetDSPProperty(hBoard, DSP_SCALEIN,   256*InputScaleFactor*(2^nShift));
SetDSPProperty(hBoard, DSP_SCALEOUT,  256*OutputScaleFactor*(2^nShift));

% write analog I/O offset values

SetDSPProperty(hBoard, DSP_OFFSETIN,  InputOffset);
SetDSPProperty(hBoard, DSP_OFFSETOUT, OutputOffset*(2^nShift2));

% set sampling rate control register (mode value)

SetDSPProperty(hBoard, DSP_FSMODE, FsMode);

% set actual sampling rate (in Hz)

SetDSPProperty(hBoard, DSP_FSVALUE, FsActual);

% Note:  uncomment the DSP_FRMSIZ property below to cause "userproc.c"
% to be inserted into the real-time data stream processing on the DSP
% board.  This file is included in the "DSP Source Code Interface"
% software, and is intended to serve as an entry point for user-defined
% real-time C (or .asm) code which runs on the DSP board.  This approach
% enables the built-in analog I/O interrupt and buffer processing
% on the board to run normally, initialized from MATLAB, and also keeps
% all host interface and communication routines intact.  This allows
% a focus only on the application-specific, C programming needed for
% real-time operation, and minimizes the amount of programming.  For
% more information, please see www.signalogic.com/ccode.htm.

% SetDSPProperty(hBoard, DSP_FRMSIZ, 1);
```

Waveform Generation Buffer Calculation and Download

```
% create data buffer; note that data must be scaled to be reasonable values
% for D/A converter(s)

for (n=1:BufSize)     % output buffer is 1 Hz sine wave

   buf1(n) = 10000*sin(2*pi*n*1000/FsActual);
   buf2(n) = 10000*sin(2*pi*n*1000/FsActual);

end

% pre-load DSP board memory with data buffers; both buffers must be preloaded
% for continuous operation.  Note that download depends on whether board has
% linear or Harvard memory architecture

if (MemArch == DS_GMA_VECTOR)   % board has vector memory architecture

    Status = PutMem(hBoard, DS_GM_VECTOR_DATA_X, OutBaseAddr, DS_GM_SIZE16_CVT,
                    buf1, BufSize);
    Status = PutMem(hBoard, DS_GM_VECTOR_DATA_Y, OutBaseAddr, DS_GM_SIZE16_CVT,
                    buf2, BufSize);

else  % board has linear or Harvard memory architecture

    Status = PutMem(hBoard, DS_GM_LINEAR_DATA_RT, OutBaseAddr, DS_GM_SIZE16_CVT,
                    buf1, BufSize);
    Status = PutMem(hBoard, DS_GM_LINEAR_DATA_RT, OutBaseAddr+BufSize,
                    DS_GM_SIZE16_CVT, buf2, BufSize);
end

if (Status == NULL)
    errstr = ['problem with PutMem; error code = ' num2str(GetEngErrStat(hEngine))];
    disp(errstr);
end
```

Run Board; Run One or More Processors

```
% run board/processor examples

   if (fMultiProcessor)
      RunProcessor(hBoard, dwProcList);
   else if (fMultiProcessor && fFirstProcActiveOnly) then
     RunProcessor(hBoard, 0x01);
   else if (fMultiProcessor && fAllProcActive)
     RunBoard(hBoard);
   else if (fSingleProcessor)
    RunBoard(hBoard);
```

Continuous I/O Data Buffer Processing

```
% Start board and wait for buffer to be output.  Note above that we filled
% second buffer also, to avoid "screech"--because DSP will start output
% on second buffer before we can stop the processor

% Note:  first buffer will be output as soon as board starts running; for
% continuous % output, "Buffer Ready" callback messages must be processed,
% and the RegMsgWnd() and WaitForBuffer() functions must be used

RunProcessor(hBoard, 1);  % run; processor #1 only

fCont = 10;               % output 10 buffers

while fCont > 0,

% wait to make sure first buffer has been output
  WaitForBuffer(hBoard, BufNum, NULL, DS_WFB_POLLED + DS_WFB_SYNC);

  if (MemArch == DS_GMA_VECTOR)  % board has vector memory architecture

    if (BufNum == 0)
        Status = GetMem(hBoard, DS_GM_VECTOR_DATA_X, TimBaseAddr,
                        DS_GM_SIZE16_CVT, buf_plot, BufSize);
    else
        Status = GetMem(hBoard, DS_GM_VECTOR_DATA_Y, TimBaseAddr,
                        DS_GM_SIZE16_CVT, buf_plot, BufSize);
    end
  else  % board has linear or Harvard memory architecture

    Status = GetMem(hBoard, DS_GM_LINEAR_DATA_RT, TimBaseAddr+BufNum*BufSize,
                    DS_GM_SIZE16_CVT, buf_plot, BufSize);
  end

  buf = buf_plot;  % store acquired data in variable buf for PutMem use

  if (Status == NULL)
    errstr = ['problem with GetMem; error code = ' num2str(GetEngErrStat(hEngine))];
    disp(errstr);
  end

% To change the output during data acquisition, uncomment the following section
% of code, which feeds the input to the output directly for each buffer
%
%  if (MemArch == DS_GMA_VECTOR)  % board has vector memory architecture
%
%    if (BufNum == 0)
%       Status = PutMem(hBoard, DS_GM_VECTOR_DATA_X, OutBaseAddr,
                        DS_GM_SIZE16_CVT, buf, BufSize);
%    else
%       Status = PutMem(hBoard, DS_GM_VECTOR_DATA_Y, OutBaseAddr,
                        DS_GM_SIZE16_CVT, buf, BufSize);
%    end
%    else  % board has linear or Harvard memory architecture
%
%    Status = PutMem(hBoard, DS_GM_LINEAR_DATA_RT, OutBaseAddr+BufNum*BufSize,
                     DS_GM_SIZE16_CVT, buf, BufSize);
%  end
%
%  if (Status == NULL)
%    errstr = ['problem with GetMem; error code = ' num2str(GetEngErrStat(hEngine))];
%    disp(errstr);
%  end

  BufNum = 1 - BufNum;  % toggle buffer number

  fCont = fCont -1;     % to loop forever, do not decrement fCont

end
```

```
% display data

ymax=max(buf_plot);          % set y-axis maximun
ymin=min(buf_plot);          % set y-axis minimum

figure(gcf);                 % use current figure, if one

axis([0, BufSize-1, ymin, ymax]);
pt = min(BufSize, 1000);
plot(buf_plot(1:pt));        % show part of the data
```

Cleanup (Board Shutdown)

```
% done; leave board in disabled state, clean up handles, engines

if (hBoard ~= NULL)

   if (fBoardInitialized == TRUE)
      DisableBoard(hBoard);
   end

   FreeBoard(hBoard);
end

if (hEngine ~= NULL)
   EngineClose(hEngine);
end
```

5.2.4.2.4 Visual Basic Examples in DirectDSP

The Visual Basic code excerpts below show examples of:

- ✓ Hardware Manager dialog box display
- ✓ Driver and board initialization
- ✓ DSP program file download
- ✓ DSP property initialization
- ✓ Run board; run one or more processors
- ✓ Buffer-ready callback message and I/O data buffer processing
- ✓ Waveform data display
- ✓ Cleanup (board shutdown)

For complete Visual Basic source code examples, see the following examples included with DSPower software:

Source Code Example Program Type	**Filename**
Debugger example	sdebug.bas
Digital oscilloscope	dscope.bas
Digital tape recorder	dtape.bas
Spectrum analyzer	spcana32.bas
Strip-chart recorder	strip.bas

Hardware Manager Dialog Box

```
' show hardware manager dialog (Hardware Manager); returns board designator
' string
' note: if hardware choice is fixed, then skip this call and use correct board
' designator string in DSAssignBoard call below

   If DSShowHardwareSelectorDlg(0, szBoard) = IDCANCEL Then
      GoTo Abort
   End If
```

Driver and Board Initialization

```
' first try to open Hypersignal-Macro or Hypersignal-Macro EX as engine

hEngine = DSEngineOpen(ByVal DS_EO_HSM, ByVal 0&, NUL)

If (hEngine = NUL) Then

' if that doesn't work, try Hypersignal-Acoustic

      hEngine = DSEngineOpen(ByVal DS_EO_HSA, ByVal 0&, NUL)

      If hEngine = 0 Then

         nStatus = DSGetEngMgrErrorStatus()
         If (nStatus = NUL) Then nStatus = DSGetHWLibErrorStatus()
         tmpstr = Str$(nStatus)
         tmpstr = "DSEngineOpen error code =" + tmpstr
         ret = DSMessageBox(NUL, tmpstr, szApp,
                            MB_ICONEXCLAMATION Or MB_ALWAYSONTOP)
         GoTo Abort

      End If

   End If

' note:  using a string name instead of the pre-defined “DS_EO_xxx” constants
' above will open a user-defined driver.  Drivers must export certain
' baseline functions required by DSPower software.  Please ask Signalogic
' for the “OEM DSPower Software Development Guide” document.

' assign a board handle:  engine handle, board designator, bus type, IO base
' addr, Mem base addr

   hBoard = DSAssignBoard(hEngine, ByVal szBoard, 0, 0, 0)

' initialize the board; make sure it's installed, reset all processors

   fBoardInitialized = DSInitBoard(hBoard)

   If (fBoardInitialized = NUL) Then

      nStatus = DSGetHWLibErrorStatus()
      If (nStatus = NUL) Then nStatus = DSGetEngineErrorStatus(hEngine)
      tmpstr = Str$(nStatus)
      tmpstr = "DSEngineOpen error code =" + tmpstr
      ret = DSMessageBox(NUL, tmpstr, szApp,
                         MB_ICONEXCLAMATION Or MB_ALWAYSONTOP)
      GoTo Abort

   End If

' interrogate engine for board type values
```

```
   wBoardClass = DSGetBoardClass(hBoard)

' get memory architecture

   uMemArch = DSGetMemArch(hBoard)

   If (uMemArch = NUL) Then

      ret = DSMessageBox(NUL, "DSGetMemArch failed", szApp,
                         MB_ICONEXCLAMATION Or MB_ALWAYSONTOP)
      GoTo Abort
   End If

' get memory size, (note that this currently has to be done after LoadFile)

   dwMemSize = DSGetMemSize(hBoard, &H1)  'processor 0 only


' reset DSP board (should already be in reset state; processor 0 only)

   ret = DSResetProcessor(hBoard, &H1)


' set the message blaster (MSGBLAST.OCX or MSGBLAST.VBX) to capture buffer ready
' messages from the DSP driver/engine

   Buffer_Ready.hWndTarget = graph_window.hWnd
   Buffer_Ready.AddMessage WM_DSPENGINE_BUFRDY, POSTPROCESS

' register handle of graph window with engine, to receive "buffer ready"
' messages; this is same as registering callback function in C/C++

   ret = DSRegisterEngineMsgWnd(hEngine, DS_REMW_SETDSPDATARDYMSG,
                                graph_window.hWnd)
```

DSP Program File Download

```
' load executable DSP code file (usually a COFF file produced by DSP
' manufacturer's linker)

' load default file for the board type (processor 0 only)

   ret = DSLoadFileProcessor(hBoard, ByVal 0&, &H1)

   If (ret = NUL) Then
      ret = DSMessageBox(NUL, "DSLoadFileBoard: problem loading file", szApp,
                         MB_ICONEXCLAMATION Or MB_ALWAYSONTOP)
      GoTo Abort
   End If
```

DSP Property Initialization

The DSP property initialization examples below may not be complete for all pre-defined operating modes in the DSP Source Code Interface. Please also consult the C/C++ and MATLAB examples for possible additional property examples.

```
' write DSP properties

   ret = DSSetDSPProperty(hBoard, DSP_BOARDCLASS, wBoardClass And &HFF)

   ret = DSSetDSPProperty(hBoard, DSP_BOARDSUBCLASS, Shift(wBoardClass, -8))

   ret = DSSetDSPProperty(hBoard, DSP_OPMODE, 5)  ' spectrum analyzer is mode 5

   If (DSGetBoardInfo(hBoard, DS_GBI_DSPWORDLENGTH) = 24) Then
```

```
      nShift = 7    ' weird efficiency thing required by DSP56xxx boards
   Else
      nShift = 0
   End If

   ret = DSSetDSPProperty(hBoard, DSP_SCALEIN, Shift(256&, nShift))

   ret = DSSetDSPProperty(hBoard, DSP_FILTTYPE1, 0) ' trace 1 real-time filt off

   ret = DSSetDSPProperty(hBoard, DSP_FILTTYPE2, 0) ' trace 2 real-time filt off

   ret = DSSetDSPProperty(hBoard, DSP_TRIGLEVEL, 0) ' free-run triggering

   ret = DSSetDSPProperty(hBoard, DSP_TRIGCHANLIST, 0)

   ret = DSSetDSPProperty(hBoard, DSP_NUMCHAN, NumChan)    ' number of channels

   ret = DSSetDSPProperty(hBoard, DSP_HOSTBUFNUM, 0)  ' starting buffer number

   ret = DSSetDSPProperty(hBoard, DSP_BUFNUM, 0)

   dw = 0
   For i = 0 To (NumChan - 1)
      dw = dw Or Shift(i, 4 * i) ' shift left
      ChanList(i) = i
   Next i

   ret = DSSetDSPProperty(hBoard, DSP_CHANLIST, dw)        ' channel list

   dwFsMode = DSCalcSampFreq(hBoard,
                             FsDesired,
                             NumChan,
                             ChanList(0),
                             dwFsActual)

   ret = DSSetDSPProperty(hBoard, DSP_FSMODE, dwFsMode)  ' sampling rate control
                                                         ' register (mode value)

   ret = DSSetDSPProperty(hBoard, DSP_GAINLIST, 0) ' gain list

   ret = DSSetDSPProperty(hBoard, DSP_FSVALUE, dwFsActual)  ' actual sampling
                                                            ' rate (in Hz)

' Framesize property required for real-time C code data transfer to host
'  operation (mode 6), or Spectrum Analyzer operation (mode 5)

   ret = DSSetDSPProperty(hBoard, DSP_FRMSIZ, Framesize)

   ret = DSSetDSPProperty(hBoard, DSP_BUFLEN, Framesize * NumChan) ' buffer size

   dw1 = 0
   dw2 = 0

   ret = DSSetDSPProperty(hBoard, DSP_MAXVAL1, dw1)

   ret = DSSetDSPProperty(hBoard, DSP_MAXVAL2, dw2)

   ret = DSSetDSPProperty(hBoard, DSP_OVERLAP, CLng(txtOverLap.Text))  ' FFT
                                                                       ' overlap

' properties below only required for Spectrum Analyzer operation (mode 5)

   ret = DSSetDSPProperty(hBoard, DSP_FFTLEN, FFTsize)

   ret = DSSetDSPProperty(hBoard, DSP_FFTORD, CLng(Log(FFTsize) / Log(2) + 0.1))

   ret = DSSetDSPProperty(hBoard, DSP_LOGFLG1, CInt(LogMagDisplay))
   ret = DSSetDSPProperty(hBoard, DSP_LOGFLG2, CInt(LogMagDisplay))

   ret = DSSetDSPProperty(hBoard, DSP_PHZREQ, CInt(PhaseDisplay))
```

```
   ret = DSSetDSPProperty(hBoard, DSP_PWRFLG1, 0)
   ret = DSSetDSPProperty(hBoard, DSP_PWRFLG2, 0)
   ret = DSSetDSPProperty(hBoard, DSP_XFERFLG1, 0)
   ret = DSSetDSPProperty(hBoard, DSP_XFERFLG2, 0)
   ret = DSSetDSPProperty(hBoard, DSP_DUPFLG, 0)

   LogCoeffA = 20# * 32767 / 100
   LogCoeffB = 32767# - LogCoeffA * (Log(32767) / Log(10))

   ret = DSIEEEToDSP(hBoard, DS_DTI_IEEESIZE32, LogCoeffA, DSPVal, 1)

   ret = DSSetDSPProperty(hBoard, DSP_LOGCOEFFA, DSPVal)

   ret = DSIEEEToDSP(hBoard, DS_DTI_IEEESIZE32, LogCoeffB, DSPVal, 1)

   ret = DSSetDSPProperty(hBoard, DSP_LOGCOEFFB, DSPVal)
```

Run Board; Run One or More Processors

```
' run board/processor examples

    If (fMultiProcessor) Then
       DSRunProcessor(hBoard, dwProcList)
   ElseIf (fMultiProcessor And fFirstProcActiveOnly) Then
     DSRunProcessor(hBoard, &H01)
   ElseIf (fMultiProcessor And fAllProcActive) Then
     DSRunBoard(hBoard)
   ElseIf (fSingleProcessor) Then
    DSRunBoard(hBoard)
   End If
```

Buffer-Ready Callback Message and I/O Data Buffer Processing

```
Sub Buffer_Ready_Message(ByVal hWnd As Long, ByVal MsgVal As Long, wParam As Long, lParam As
Long, nPassage As Integer, lReturnVal As Long)


Dim uStatus As Integer
Dim ret As Integer


   DoEvents  ‘ needed in Win16 VB3/4 only

   If (MsgVal = WM_DSPENGINE_BUFRDY) Then

      If (Not fRunning) Then  ' board already stopped
          Exit Sub
      End If

      If (hBoard = NUL) Then  ' close is in progres
         Exit Sub
      End If

    ' transfer below illustrates two common type of DSP memory architectures

      If (LinMagDisplay Or LogMagDisplay Or PhaseDisplay) Then

         If (uMemArch = DS_GMA_VECTOR) Then  ' vector data memory

          ' magnitude data always in X: memory

            uStatus = DSGetMem(hBoard, DS_GM_VECTOR_DATA_X,
                               dwMagBufferBaseAddr, DS_GM_SIZE16,
                               FFTBuffer(0), 2 * FFTsize * NumChan)

         Else  ' magnitude data not double-buffered

            uStatus = DSGetMem(hBoard, DS_GM_LINEAR_DATA_RT,
```

```
                              dwMagBufferBaseAddr, DS_GM_SIZE16,
                              FFTBuffer(0), 2 * FFTsize * NumChan)

         End If

      ElseIf (TimeDisplay) Then

         If (uMemArch = DS_GMA_VECTOR) Then  ' vector data memory

            If (nCurBuf = 0) Then

               uStatus = DSGetMem(hBoard, DS_GM_VECTOR_DATA_X,
                                  dwTimBufferBaseAddr, DS_GM_SIZE16,
                                  Buffer(0), Framesize * NumChan)

            Else

               uStatus = DSGetMem(hBoard, DS_GM_VECTOR_DATA_Y,
                                  dwTimBufferBaseAddr, DS_GM_SIZE16,
                                  Buffer(0), Framesize * NumChan)

            End If

         Else  ' linear data/prog memory, or modified harvard arch.
               ' with linear data memory

            uStatus = DSGetMem(hBoard, DS_GM_LINEAR_DATA_RT,
                               dwTimBufferBaseAddr + nCurBuf * Framesize *
                               NumChan, DS_GM_SIZE16, Buffer(0), Framesize *
                               NumChan)

         End If
      End If


      If (uStatus = NUL) Then
         ret = DSMessageBox(Spcana32.hWnd, "DSGetMem:  problem with point
                            transfer", szApp, MB_ICONEXCLAMATION)
      End If

      nCurBuf = nCurBuf Xor 1  ' switch buffers

      ret = DSSetDSPProperty(hBoard, DSP_HOSTBUFNUM, nCurBuf)  ' tell DSP about
                                                                ' buffer switch

    ' instruct engine to wait for next buffer
    ' note this is deferred until after data display to avoid Close Event
    ' problem in VB32

      ret = DSWaitForBuffer(hBoard, nCurBuf, 0&, DS_WFB_POLLED)


    ' example of plotting input data points

      fDataRcvd = True
      If (fFirstBuffer = 0) Then
        fFirstBuffer = 1              ' first buffer received
      Else
        fFirstBuffer = 2              ' successive buffer received
      End If

      DisplayBufferData(Buffer)

   End If

   DoEvents  ‘ Win16 only

End Sub
```

Waveform Data Display

The example below uses Microsoft's graph32.ocx control to show how to display different types of waveform data transferred from DSP memory. Note that multichannel data is not shown, but is easy to incorporate. Multichannel data is interleaved in Number-of-Channel groups; for example, for 4-channel data:

ch0,ch1,ch2,ch3,ch0,ch1,ch2,ch3, ...

```
Private Sub DisplayBufferData(Buffer As Integer*)

Dim i As Long
Dim j As Integer
Dim Data As Integer          ' temporary storage for data in buffer
Dim ret, inc, index As Integer

' update the scope window with data currently in Buffer

   graph_window.NumSets = NumChan
   graph_window.ThisSet = 1
   graph_window.ThisPoint = 1

   VOffset = 0
   Vscale = 1

' note:  current display assumes real-valued input data to FFT;
' spectrum is symmetric so only half of freq. domain data needs to be displayed

   If (LinMagDisplay Or LogMagDisplay Or PhaseDisplay) Then
      graph_window.NumPoints = FFTsize / NumChan / 2
      inc = 2
   Else
      graph_window.NumPoints = Framesize / NumChan
      inc = 1
   End If

   If (PhaseDisplay Or LogMagDisplay) Then
      index = 1
   Else
      index = 0
   End If

   graph_window.ThisSet = 1

   For i = 0 To graph_window.NumPoints - 1

      If (LinMagDisplay Or LogMagDisplay Or PhaseDisplay) Then
         Data = FFTBuffer(index)
      ElseIf (TimeDisplay) Then
         Data = Buffer(index)
      End If

      graph_window.GraphData = Data * VScale + VOffset

      index = index + inc

   Next

   For j = 1 To graph_window.NumSets

      graph_window.ThisSet = j
      graph_window.ColorData = j ' Set different channel to different color
      graph_window.DrawMode = 3  ' Blt to screen, to avoid flicker
   Next

End Sub
```

Cleanup (Board Shutdown)

```
Private Sub Form_Unload(Cancel As Integer)

Dim ret As Long


' free board, close engine if needed

   If (hBoard <> NUL) Then

      If (fBoardInitialized <> 0) Then
         ret = DSDisableBoard(hBoard)
      End If

      ret = DSFreeBoard(hBoard)
      hBoard = NUL

   End If

   If (hEngine <> NUL) Then
      ret = DSEngineClose(hEngine)
      hEngine = NUL
   End If

   End  ' shut-down application

End Sub
```

5.2.5 C67xx Source Code Interface Structure

Working with DSP Source Code Interface software for Hypersignal is very straightforward. The fact that there exists similar implementations for nine (9) different DSP chip families has led to standardization of library calls and operational procedures, from the function level down to the batch files used to facilitate the code modification process. For example, all DSP chip families have similar routines available for post-FFT processing, such as log, square root, arctan, etc. And, at a lower level, all "LNK.BAT" files produce a map file of the form "xxxLINK.MAP" where xxx is “T54” for Texas Instruments C67xx, "T30" for Texas Instruments C3x, "M56" for Motorola 56xxx, “A60” for Analog Devices 2106x, etc. Moreover, the modification and download process is also standardized between DSP families. For example, Hypersignal parses and downloads output files from each DSP manufacturer's linker to the DSP board for subsequent execution, obviating the need for user-written object file downloaders.

All interrupt service routines fully save and restore machine context to protect registers they modify and some routines (such as the “snap-in” real-time filter routines included in the C67xx source code library also save and restore registers they use. In all chip families the Digital Oscilloscope and Spectrum Analyzer functions provide the best examples of full context-save interrupt service routines combined with substantial foreground processing constructed from many of the routines in the source code library.

The C67xx Source Code Interface base file list includes .asm and .c function and analog I/O files plus linker control files and support utilities.

The C67xx source code interface supports the basic "modes" of operation below:

C67xx .Asm File	Function	operating mode
MODUL0	Analog Conversion	mode=0
MODUL1	Analog Conversion, Playback from Waveform Display	mode=1
MODUL2	Digital Oscilloscope	mode=2
MODUL3	FFT file-based acceleration	mode=3
MODUL4	iFFT file-based acceleration	mode=4
MODUL5	Spectrum Analyzer	mode=5
MODUL6	Stimulus & Response operation	mode=6
MODUL7	Convolution/Correlation file-based acceleration	mode=7
MODULn	User-Defined function	mode=n

5.2.5.1 C67xx Source Code Interface Interrupt Service Routines

There are various interrupt service routines that respond to interrupts on the C67xx boards and generally match up with the foreground processing modules above. Interrupt service routine module names consist of "ISRn", where n indicates the operating mode (see section 5.1.5 above) that they support, for example "ISR0". Inside each interrupt service routine there are different entry points that contain hardware-specific input/output code for individual boards. These hardware-specific sections are kept as small as possible, so that the great majority of the routine is common for all boards. Examine the interrupt vector and mask initialization in the initial setup code in "T67BEG", depending on board type (see host-shared variable "BOARD"), to locate the correct interrupt service routine for a particular function.

5.2.5.2 C67xx Source Code Interface Host PC Communication

It is important to keep in mind the resident, interrupt-driven interface between host software and the C67xx code. A very small “talker” program is used on the DSP-side to allow two-way communication between the host PC and C67xx DSP to be established. Certain scratch memory locations used by the talker, and the program code area itself used by the talker, absolutely must not be changed in order to maintain high-level host software communication. For example, application-specific DSP code should be linked such that new code does not “step on” the talker;

one way to do this is to leave an non-initialized section, or hole, in the linker command file. An example linker command file is printed below which shows how to do this.

In addition, there are several pre-defined “DSP properties” which can be set by the host to initialize and control default operating modes. These properties are implemented as shared, global variables in the DSP code; they are well documented in the file t54beg.asm. *Do not insert any code of any type before these addresses that would cause them to move*.

The talker currently resides in onchip program memory, from 0x80 to 0x100. In addition, it uses 6 words in onchip memory at 0x1000 for a “command structure” used to communicate with host drivers.

See Appendix B, C67xx Talker Program, for talker source code and map file.

5.2.5.3 C67xx Source Code Interface Buffers and Real-Time Streaming

Currently, operating mode 5 (Spectrum Analyzer) is the best example of an extensive foreground processing combined with context-save background interrupts. The Spectrum Analyzer source code represents a totally real-time system, with samples buffered in a dual-buffer scheme in response to background "sample ready" interrupts, and buffers, when full, sent to FFT and post-FFT processing in the foreground. Use mode 5 source modules ("MODUL5" and modules it calls), and "ISR5" for one example of a real-time, interrupt-driven structure.

5.2.6 C67xx Source Code File List

A partial list of source code and related files included in the C67xx DSP Source Code Interface software follows:

T67DEF.ASM	Variable & constant definition
T67BEG.ASM	Beginning vectors, including ISR vectors, and including initialization, including PC-written variables
ISR0.C	Continuous A/D ISR for all board
ISR1.C	Continuous D/A ISR for all boards
ISR2.C	Digital Scope ISR for all boards.
ISR5.C	Spectrum Analyzer ISR for all boards.
ISR6.C	Stimulus & Response ISR for all boards.
CODEC.C	Misc. codec initialization algorithms
C6XXXLIB.C	C6xxx SCI library functions

FFT2.C	Radix-2 FFT algorithm
MODUL0.C	Foreground A/D-to-disk processing (pre-defined mode=0)
MODUL1.C	Foreground D/A-from-disk processing (pre-defined mode=1)
MODUL2.C	Foreground Digital Oscilloscope processing (pre-defined mode=2)
MODUL3.C	FFT file-based acceleration processing (pre-defined mode=3)
MODUL4.C	iFFT file-based acceleration processing (pre-defined mode=4)
MODUL5.C	Foreground Spectrum Analyzer process (pre-defined mode=5)
MODUL6.C	Foreground stimulus & response (pre-defined mode=6)
MODUL7.C	Convolution/correlation file-based acceleration processing (pre-defined mode=7)
VECTORS.ASM	Vector table definitions
SINTAB.ASM	Sine value lookup-table for FFT use
USERPROC.C	Example UserProc entry-point function
C6XXXLIB.H	Include file for C6xxxlib.c
CODEC.H	Include file for codec.c
C6X.H	Miscellaneous definitions for C6xxx processors
C6211DSK.H	Miscellaneous definitions for C6xxx processors
LNK.BAT	Batch file used to link all source modules: type "LNK"
T67LINK.CMD	Link control file (batch file for linker)

The distribution software does not initially contain all .OBJ files necessary to produce a complete, default executable code module when the T54LINK.CMD control file (above) is invoked with the Texas Instruments linker utility. Each .ASM module above must be assembled first; use the BUILD.BAT file to do this.

5.2.7 C67xx Source Code Modification Process

The steps taken to modify, add and integrate new source code are listed below:

1) Make changes to C67xx source code, keeping in mind the necessary shared variables between Hypersignal and the C67xx processor. The file T67DEF.ASM is well documented in this area.

2) Assemble all modules changed and any new modules created (using Texas Instruments "COFF" assembler) to ensure that a new .OBJ (object) file is generated for each altered module. For convenience, an "ASM.BAT" batch file has been provided that can be invoked as

```
ASM filename <car ret>
```

to assemble source code modules (entering the .ASM extension after "filename" is optional). This file executes

```
asm500 -l -v549 -mg %1 > %1.err
type %1.err
```

statements, where %1 is the filename. *If a new module has been created, be sure to add it to the T54LINK.CMD control file* (use an ASCII text-editor to update the T54LINK.CMD file).

3) Re-link all modules using Texas Instruments C67xx "lnk500" linker. A "LNK.BAT" batch file has been provided that can be used by invoking:

```
LNK <car ret>
```

to link all necessary .OBJ files. LNK.BAT executes a

```
lnk500 T54LINK.CMD
```

statement, where "T54LINK.CMD" is an ASCII linker-command file that specifies all .OBJ files to be linked by the dsplnk.exe program, as well as specific information about memory addresses for some sections.

4) Copy the resulting .OUT file to \hsmacro or \dspower subdirectories, run the desired host program, and invoke the appropriate functions. When using DSPower programs, the .OUT filename can be specified in the Hardware Manager dialog box (DSP Program File field) or in a LoadFileProcessor() or LoadFileBoard() function call. When using Hypersignal software, the .OUT filename can be specified in the System Config menu (added after the board designator with a separating comma). It is also possible to simply copy the resulting .OUT file to \hsmacro or \dspower subdirectories over the existing, default tmsc54x.out file. However, in this case, it is suggested that a backup of the original file be made first in case the new code does not work. In either case, the DSPower or Hypersignal COFF file download function calls will read the .OUT file directly according to Texas Instruments COFF file format and download all code and data. A subsequent RunProcessor() or RunBoard() function call will begin execution at the reset vector location.

5) There are several references to "trace" and "channel" in DSP source code and source code variable descriptions. Traces are references to waveforms in displays or waveform files, whereas channels refer to the physical analog input or output connection on the DSP/analog hardware. A logical-to-physical mapping takes place when channel lists are sent from Hypersignal software or DSPower programs to executing DSP code.

5.2.7.1 C67xx Source Code Interface Modification Notes

Some notes about C67xx source code modification:

1) All code has been tested using version 2.1 of Code Composer Studio.

2) To modify the default time data, frequency domain output data, or other buffer addresses, please consult the comments and suggestions in the T67DEF.ASM file.

3) The Code Composer Studio linking procedure produces a map file (extension .MAP) which can be used to check for overlaps and current locations of variables, buffers, and code sections. The map file also shows the starting address and length of each code module included in the current executable output file. When significant amounts of code are added, extra attention should be paid to the length of the executable file as compared to the amount of program memory available. If the available program memory is exceeded, problems can occur which may be difficult to pin down. Such problems can seem to result from other causes, and can lead to unnecessary time spent searching for the real cause.

4) If DSP program filename has been changed from the default (by renaming the output file in the CCS project options), DSPower programs or Hypersignal software must know about the name change. To do this in DSPower programs, change the value of the DSP Program File field in the Hardware Manager dialog box (or change the value of LoadFileProcessor and LoadFileBoard calls in the programs). To do this in Hypersignal software, suffix the filename to the appropriate field in the System Config menu. For example, if the Hypersignal Analog Conversion function is being used for test, then add the new filename to the ANALOG CONVERSION field in the System Config menu; for example, this might be an entry like 'IIC31-B-50,test.out', 'TBS56-B,AC3.OUT' or 'sig32c,dsp32cxx.out'.

5) To use an external sample clock, follow the basic guidelines below:

- Set any jumpers that may be required on the board or module itself. See the jumper list and/or hardware manufacturer documentation.
- Select External Sample Clock in the Hardware Manager dialog box or in DSPower function calls, or enter '-EXT' after the board designator in the Hypersignal System Configuration menu. This causes the high-level software to set the value of the shared variable "FS_MODE" as appropriate for external clock operation. If due to the design of the board no setting is possible, then the value of FS_MODE will be zero. If the high-level software does not set the value correctly, or the hardware has customized external

sampling circuitry, then the FS_MODE variable should still be used to control the sampling rate. In this case, the DSP source code should be modified as needed.

- The actual sampling rate value is contained in the shared variable "FS_VALUE". The value is in Hz (Hertz); note that for C67xx boards, which use 16-bit wide memory accesses, the maximum value that can be represented is limited to 65535. This value is initialized by the high-level software, based on what is entered in the current menu (SAMPLING FREQ field).

- Enter sampling rate values in DSPower demo or application programs, or SAMPLING FREQ field values in Hypersignal software menus, to match the actual external sample clock rate. This will ensure that time and frequency unit readouts in all real-time and post-acquisition displays are accurate.

5.3 Interface from Application C Code

The method described here is more general than the “UserProc entry-point” method described in section 5.1.3.1 above, UserProc: Real-Time ‘C’which focuses on inserting specific C code and routines into an existing real-time buffer stream. The method described in this section also handles buffer management and so takes on another layer of responsibility, which in the “UserProc” method was assumed to be handled by the framework.

The real-time C code example below is based on a MELP algorithm that has two (2) basic components: encode and decode. The code shown is a simplified “void main()” which does the following:

- performs initialization and setup
- waits for an analog I/O buffer to be ready
- manipulates input data
- calls MELP analysis and/or synthesis
- manipulates output data
- loops to wait for the next buffer; this is a perpetual loop and continues until the processor is reset by the host system

Here are some notes about the example code:

1) The bzyflg variable is being controlled by the framework; specifically, the isr6.asm file in the C67xx Source Code Interface. Each time bzyflg changes value (i.e. toggles from 0 to 1 and vice versa) and new buffer is available. This means:

   - a new buffer of sampled time data has been gathered by the background ISR and analog input hardware drivers and is ready for processing

   - the last buffer of processed data has been output by the analog output drivers

2) The routines getData, sendData, getCoeff, sendCoeff are not shown for clarity. These are in the .c files included in the MELP source code.
3) In this example, the actual routines which call analysis (encode) and synthesis (decode) are melp_ana and melp_syn.

```
/*****************************************************************************
/*
/*        COPYRIGHT (C) SIGNALOGIC, INC, 1991-2000
/*
/*        Mixed Excitation LPC speech coder
/*        void main()
/*        initialization
/*        foreground loop
/*        buffer processing
/*
*****************************************************************************/


/* compiler include files  */

#include "melp.h"
#include "spbstd.h"
#include "mat.h"

/* Globals */
int saturation = 0;
int complexity;

/* mode constants (can be OR'ed together) */
#define ANALYSIS 1
#define SYNTHESIS 2
#define REAL_TIME 1
#define SIMULATION 2

#define FRAMESIZE 180   /* size of input frame (time samples) */
#define CHSIZE 9        /* number of 6-bit coefficients per input frame */
#define NUM_CH_BITS 54  /* bits per frame */

/* external memory */
Shortword melpmode;
Shortword opmode;

Shortword speech_in[FRAME];
Shortword speech_out[FRAME];
Shortword speech_save[FRAME];

Shortword loopcnt;

UShortword chbuf[CHSIZE];

volatile Shortword bzyflg;
Shortword melpflg;

long int getframeaddr;
long int sendframeaddr;
long int getcoeffaddr;
long int sendcoeffaddr;

long int rdmemptr;
long int wrmemptr;
long int rdcoeffptr;
long int wrcoeffptr;

Shortword TxByteFlag;
Shortword RxByteFlag;

Shortword RxPTR;
```

```
Shortword RxBP;
Shortword RxMP;
Shortword RxEP;

Shortword TxPTR;
Shortword TxBP;
Shortword TxEP;



void main() {

 /* initialize variables */

    rdmemptr = getframeaddr;
    wrmemptr = sendframeaddr;
    rdcoeffptr = getcoeffaddr;
    wrcoeffptr = sendcoeffaddr;

 /* initialize serial port variables */

    RxBP = (Shortword)getframeaddr;
    RxPTR = RxBP;
    RxMP = RxBP + FRAMESIZE;
    RxEP = RxBP + 2 * FRAMESIZE;


    TxBP = (Shortword)sendframeaddr;
    TxPTR = TxBP;
    TxEP = TxBP + 2 * FRAMESIZE;


 /* re-initialize sys_memory */

    minit();

 /* Initialize MELP analysis and synthesis */

    if (melpmode != SYNTHESIS)
      melp_ana_init();

    if (melpmode != ANALYSIS)
      melp_syn_init();

    if (opmode == REAL_TIME) {

        bzyflg = 0;
        melpflg = 0;

        BSP_Init();

        melp_real();
    }
    else
        melp_simu();
}


/* Real-Time operation:  run MELP coder on analog input signal buffers */

int melp_real() {

Shortword i,j;
Longword frame;
Shortword eof_reached;
static struct melp_param melp_par;    /* melp parameters */
Shortword *input, *output, temp;

    frame = 0;
    melp_par.chptr = chbuf;
    melp_par.chbit = 0;
```

```
    eof_reached = 0;

    loopcnt = 0;

    while (eof_reached == 0) {

        while (bzyflg == melpflg) {};  /* wait for buffer-ready from ISR */

#if debug

        getData(speech_in, FRAME, melpflg);

        for (i=0; i<FRAME; i++) speech_out[i] = speech_in[i];

        sendData(speech_out, FRAME, melpflg);

#elif

     /* if Analysis enabled... */

        if ((melpmode & ANALYSIS) != 0) {

        /* get data from analog/digital input buffer */

           getData(speech_in, FRAME, melpflg);

            if ((melpmode & SYNTHESIS) != 0) {  /* if Synthesis enabled also */

                melp_par.chptr = chbuf;  /* reset pointers to channel buffer */
                melp_par.chbit = 0;
            }


            melp_ana(speech_in, &melp_par);  /* call MELP Analysis */


            if ((melpmode & SYNTHESIS) == 0) {  /* if analysis only */

                sendCoeff(chbuf, CHSIZE, 0);  /* send bit stream to the PC */

                if (melp_par.chbit == 0) {  /* write channel output if needed */

                    melp_par.chptr = chbuf;  /* reset ptr to channel buffer */
                }
            }
        }

     /* if Synthesis enabled... */

        if ((melpmode & SYNTHESIS) != 0) {

            if (melpmode & ANALYSIS) == 0) {  /* if no Analysis... */

                getCoeff(chbuf, CHSIZE);  /* read channel input */
            }

            melp_par.chptr = chbuf;  /* reset pointer to channel buffer */
            melp_par.chbit = 0;

            melp_syn(&melp_par, speech_out);

         /* send synthesis results to analog/digital output buffer */

            sendData(speech_out, FRAME, melpflg);
        }

        loopcnt++;  /* bump loop counter */

        frame++;  /* bump frame counter (used by host PC to measure MIPs) */

#endif
```

```
        melpflg = 1 - melpflg;  /* toggle buffer indicator */
    }
}
```

5.4 MELP Source Code Software

For a list of MELP source code files, please see the melp.cmd file listed in Appendix C below, MELP Linker Command and Map Files.

5.4.1 Modifying MELP Source Code

The process for modifying MELP source code is straightforward. Two methods are documented below:

- command-line
- Code Composer Studio

Command-Line MELP Source Modification

The steps required to modify MELP source code and re-build a MELP real-time executable COFF file are listed below:

1) Modify C or asm files using text-editor.

2) Run Asm batch (command) file for asm files that use mnemonic instruction set. Run Asma.bat file for asm files that use algebraic instruction set.

3) Run C54 batch (command) file for C files.

4) Run Lnk batch (command) file to link all .obj files and re-build a MELP executable COFF file (default filename is melptest.out).

5) Copy melptest.out (or other linker output) file to \melp\dsp subdir.

6) Re-run MELP IDE software; make sure DSP Program File field in Hardware Manager dialog box matches the linker output filename used in step 5) above.

7) Repeat above steps as needed.

Code Composer Studio MELP Source Modification

This section not added yet.

5.4.1.1 MELP Source Code Modification Notes

Below are some important notes to keep in mind when modifying MELP source code:

1) The default melptest.out executable has been built with the following Texas Instruments development tools:

Assembler	TMS320C54x COFF Assembler, v3.10
Linker	TMS320C54x COFF Linker, v3.10
Compiler	TMS320C53x C Compiler ShellVersion, v3.10

2) The default Asm.bat file command-line is:

```
asm500 –g –l –v548 %1 > %1.err
type %1.err
```

The default Asma.bat file command-line is:

```
asm500 –mg –l –v548 %1 > %1.err
type %1.err
```

The default C54.bat file command-line is:

```
cl500 –g –pk –ss –al –v548 %1 %2 %3
```

The default Lnk.bat file command-line is:

```
lnk500 melp.cmd
```

5.5 MP3 Source Code Software

This section not added yet.

Appendix A C54xx Source Code License Agreement

The standard C54xx Source Code License Agreement is re-printed below. Note a signed Non-Disclosure Agreement (NDA) is also required prior to obtaining C54xx Source Code Interface software. Basically, the NDA requires the C54xx Source Code to be used for in-house R&D and product-development purposes, and prohibits use of C54xx Source Code in developing a product that competes with Signalogic DSP Software components. To receive a copy of the NDA, please call Signalogic at 1-800-DSPower and provide a fax number.

Note: Certain algorithm source code products that incorporate the C54xx Source Code Interface (e.g. MELP, MP3, etc.) may supercede or augment either the C54xx Source Code License Agreement or the NDA requirement or both. Currently, this includes:

Product	**License Agreement Status**
MELP C54xx Source Code	MELP Source Code License Agreement Supercedes All
MP3 C54xx Source Code	MP3 Source Code License Agreement Supercedes All

C54xx Source Code License Agreement

Hyperception, Inc. and Signalogic, Inc. reserve all rights to C54xx Source Code supplied with "Hypersignal" series software. The C54xx Source Code may still contain sections of original C25 Source Code Developed at Hyperception in 1987-1991, and protected by License Agreements established between Hyperception and Signalogic in 1991-1993.

YOU means you, your group, company, or organization.

COMPETING PRODUCT means a product that performs similar digital signal processing and/or math processing, analysis, display, measurement, instrumentation, data acquisition, or real-time signal processing functions as Signalogic DSP software products.

THE PURPOSE OF THIS LICENSE AGREEMENT IS TO ESTABLISH THE BASIS ON WHICH YOU CAN USE THE C54XX SOURCE CODE, BUT PREVENT YOU OR ANOTHER PARTY ASSOCIATED WITH YOU FROM INCLUDING THE C54XX SOURCE CODE WITHIN OR WITH A COMPETING PRODUCT. USE OF THE C54XX SOURCE CODE BY YOU FOR PURPOSES OF RESALE OR PROFIT, OR FOR PARTIAL OR WHOLE INCLUSION IN ANY OTHER SOFTWARE INTENDED FOR SALE OR PROFIT, IS STRICTLY PROHIBITED.

You are free to copy source code sections to make backup copies and to perform experimentation and testing during algorithm and product development and research. If it is discovered that you have enabled another party to obtain C54xx Source Code through resale, lending out, carelessness, negligence, or inaction then HYPERCEPTION AND SIGNALOGIC ARE PREPARED TO PROSECUTE TO THE FULLEST EXTENT OF THE LAW. You must treat the C54xx Source Code with the same care and protection as your own valuable confidential and proprietary information.

YOU ACKNOWLEDGE THAT YOU HAVE READ THE ABOVE AGREEMENT, UNDERSTAND IT, AND AGREE TO BE BOUND BY ITS TERMS AND CONDITIONS. YOU FURTHER AGREE THAT IT AND THE HYPERSIGNAL DSP SOURCE CODE NON-DISCLOSURE AGREEMENT IS THE COMPLETE AND EXCLUSIVE STATEMENT OF ANY AGREEMENT BETWEEN US, INCLUDING ANY OTHER ORAL OR WRITTEN COMMUNICATION.

Appendix B C54xx Talker Program

Below are the source code, linker command file and map file for the C54xx talker program. See also section 5.1.5.2 above, C54xx Source Code Interface Host PC Communication.

```
; ************************************************************************
; dskc54hi.asm:  Talker program for DSPower and Hypersignal software
; Copyright (C) 1998-2002, Signalogic
;
; Created:  Oct98, RC
; Modified: Nov98, RC
; Modified: Dec98, RC
; Modified: Jun98, RC    (added far addressing)
; Modified: Jul99, JHB   (fixed 100h reset bug, added "interrupt safe" loop)
; Modified: Jan01, JHB   (added stack pointer initialization)
; Modified: Apr02, JHB   (serial port reset at program launch)
; Modified: May02, JHB   (added far code support)
; Modified: Jun24, JHB   (added BSP/McBSP 0 and 1 reset and Tx clear in launch command)
;
; ************************************************************************

         .width   80
         .length  55

         .title "TMS320C54xx Talker Program, Loaded via HPI"

         .def    main
         .def    hpiint

         .mmregs

         .bss        COMMAND,1
         .bss        PUDDADDR,1
         .bss        PUADDRHI,1
         .bss        LENGTH,1
         .bss        DUPDADDR,1
         .bss        HPIbuf,1

;=======================================================================
;
;  BEGIN OF MAIN PROGRAM
;
;=======================================================================


         .text

reset    dgoto   hi_start      ; 00: RESET
         nop
         nop

nmi      return_enable         ; 04: non-maskable external interrupt
         nop
         nop
         nop

trap2    return_enable         ; 08: trap routine (#17)
         nop
         nop
         nop
```

```
         .space 52*16          ; 0C-3F: vectors for software interrupts 18-30

int0     return_enable         ; 40: external interrupt int0
         nop
         nop
         nop

int1     return_enable         ; 44: external interrupt int1
         nop
         nop
         nop

int2     return_enable         ; 48: external interrupt int2
         nop
         nop
         nop

tint     return_enable         ; 4C: timer interrupt
         nop
         nop
         nop

brint    return_enable         ; 50: BSP receive interrupt
         nop
         nop
         nop

bxint    return_enable         ; 54: BSP transmit interrupt
         nop
         nop
         nop

trint    return_enable         ; 58: TDM receive interrupt
         nop
         nop
         nop

txint    return_enable         ; 5C: TDM transmit interrupt
         nop
         nop
         nop

int3     return_enable         ; 60: external interrupt int3
         nop
         nop
         nop

; save xpc first, before it's modified by far goto instruction

hpiint   push(xpc)             ; 64: Host-to-HPI interrupt
         far goto main
         nop

         .space  24*16         ; 68-7F: reserved area


; ******************Talker Command Block Structure ****************************
;
;   +-------+
;   | 1000h | COMMAND <-- All cases:  Command
;   +-------+
;
;   +-------+
;   | 1001h | PUDDADDR<-- Data Download & Program Upload: Source address
;   +-------+             Data Upload & Program Download: Destination address
;                         Run: Address to run
;   +-------+
;   | 1002h | PUADDRHI<-- Data Download & Program Upload: Source address (hi 16-bit)
;   +-------+             Data Upload & Program Download: Destination address (hi 16-bit)
;                         Run: Address to run (hi 16-bit)
;   +-------+
```

```
;   | 1003h | LENGTH  <-- Upload and Download: Length value
;   +-------+             Run: not used
;
;   +-------+
;   | 1004h | DUPDADDR<-- Data Download & Program Upload: Destination address
;   +-------+             Data Upload & Program Download: Source address
;                         Run: not used
;   +-------+
;   | 1005h | HPIbuf  <-- All cases: Start of HPI data transfer buffer
;   +-------+             Maximum length of each Talker transfer: 250
;
;
;***************************************************************************

         .sect  "HI_START"

hi_start:

         pmst = #00e0h         ; ovly=1; iptr=1 (*vectors= 0x80)
                               ; (note: PMST value may be changed dynamically by host)
         nop
         nop
         nop
         nop
         nop

         sp = #0x1200          ; set stack just above HPI transfer area
                               ; (can be changed by user program)

         ifr = #0xffff         ; clear pending interrupts, if any
         imr = #200h           ; enable HPIINT

         intm = #0             ; enable interrupts
wait:
         nop
         nop

         goto wait             ; initialized; wait for host-interrupts


main:    nop                   ; in case hpic = 2 caused a SAM to HOM change
         nop                   ; (non-enhanced HPI only)

         push(st0)
         push(st1)
         push(ar3)
         push(ar2)
         push(ar1)
         push(al)
         push(ah)
         push(ag)


         intm = #0             ; re-enable interrupts to allow serial port
                               ; I/O (analog I/O, TDM, etc.)

         ar1 = data(LENGTH)    ; ar1 has xfer length (host sends length - 1)
         ar3 = data(COMMAND)   ; ar3 has command value
         ar2 = data(DUPDADDR)  ; ar2 has data addr
         al = data(PUDDADDR)   ; a has prog addr (24 bit value)
         ah = data(PUADDRHI)

         if (*ar3- != 0) goto nxt1 ; cmd=0:  upload from data mem

         ar3 = a

loop0:   if (*ar1- != 0) dgoto loop0
         *ar3+ = *ar2+
         nop

         goto restore
```

```
nxt1:    if (*ar3- != 0) goto nxt2 ; cmd=1:  download to data mem

         ar3 = a

loop1:   if (*ar1- != 0) dgoto loop1
         *ar2+ = *ar3+
         nop

         goto restore


nxt2:    ar1 = #LENGTH         ; prog transfers use repeat(*ar1) form

         if (*ar3- != 0) goto nxt3 ; cmd=2:  upload from prog mem

         repeat(*ar1)          ; notes: 1) C549 has bug where prog(a) instruc-
         *ar2+ = prog(a)       ;           tion is interrupted, and then ISR
                               ;           does another prog(a), messes up
                               ;           the PAR upon interrupt return.
                               ;           MUST use repeat instruction 1-time
                               ;           ONLY to avoid this bug.
                               ;        2) interrupts are disabled during
                               ;           repeat instruction

         goto restore


nxt3:    if (*ar3- != 0) goto nxt4 ; cmd=3:  download to prog mem

         repeat(*ar1)          ; notes: 1) C549 has bug where prog(a) instruc-
         prog(a) = *ar2+       ;           tion is interrupted, and then ISR
                               ;           does another prog(a), messes up
                               ;           the PAR upon interrupt return.
                               ;           MUST use repeat instruction 1-time
                               ;           ONLY to avoid this bug.
                               ;        2) interrupts are disabled during
                               ;           repeat instruction
         goto restore


; check for command = 4

nxt4:    if (*ar3- != 0) goto nxt5 ; cmd=4:  launch program

         intm = #1             ; disable interrupts before launch

         imr = #200h           ; leave HPIINT enabled, but disable analog
                               ; I/O, any other interrupts

         ifr = #0xffff         ; clear pending interrupts, if any

         sp = #0x1200          ; reset stack (see init code above); avoid
                               ; possible case of stack overflow due to
                               ; repeated start/stop of programs

       ; set BSPn and McBSPn control registers to reset state

         bl = #0
         bh = #1

         ar3 = #0x21           ; BSP0
         *ar3+ = b
         *ar3+ = b
         *ar3 = b

         ar3 = #0x38           ; McBSP0
         *ar3+ = b
         *ar3- = b
         *ar3+ = b >> #16
```

```
         *ar3- = b

         ar3 = #0x41           ; BSP1
         *ar3+ = b
         *ar3+ = b
         *ar3 = b

         ar3 = #0x48           ; McBSP1
         *ar3+ = b
         *ar3- = b
         *ar3+ = b >> #16
         *ar3- = b


         goto a                ; launch, no return


; check for command = 5 (breakpoint)

nxt5:


restore: intm = #1             ; disable interrupts before restore

         ag = pop()
         ah = pop()
         al = pop()
         ar1 = pop()
         ar2 = pop()
         ar3 = pop()
         st1 = pop()
         st0 = pop()

         ifr = #200h           ; clear pending HPI interrupt, if any

         far dreturn_enable    ; return to user program, re-enable ints.

         hpic = #0ah           ; flag host operation complete
                               ; (note: HPIC value may be changed dynamically by host)

         .end
```

Talker Command File

```
-o DSKC54HI.OUT -m DSKC54HI.MAP

MEMORY {
         PAGE 0: PROG: ORIGIN = 00H, LENGTH = 027FFH
         PAGE 1: DATA: ORIGIN = 00H, LENGTH = 027FFH
       }

SECTIONS {
           .text        80H : { } PAGE 0
           HI_START     100H : { } PAGE 0
           .bss         1000H : { } PAGE 1
         }

DSKC54HI
```

Talker Map File

```
******************************************************************************
TMS320C54x COFF Linker           PC Version 3.70
******************************************************************************
>> Linked Wed Jul 17 10:33:30 2002
```

```
OUTPUT FILE NAME:   <DSKC54HI.OUT>
ENTRY POINT SYMBOL: 0


MEMORY CONFIGURATION

                  name            origin    length      used    attr    fill
         ----------------------  --------  ---------  --------  ----  --------
PAGE  0: PROG                    00000000   000027ff  000000fb  RWIX

PAGE  1: DATA                    00000000   000027ff  00000006  RWIX


SECTION ALLOCATION MAP

 output                                  attributes/
section   page    origin      length       input sections
--------  ----  ----------  ----------   ----------------
.text      0    00000080    00000080
                  00000080    00000080     DSKC54HI.obj (.text)

HI_START   0    00000100    0000007b
                  00000100    0000007b     DSKC54HI.obj (HI_START)

.bss       1    00001000    00000006     UNINITIALIZED
                  00001000    00000006     DSKC54HI.obj (.bss)

.data      1    00000000    00000000     UNINITIALIZED
                  00000000    00000000     DSKC54HI.obj (.data)


GLOBAL SYMBOLS: SORTED ALPHABETICALLY BY Name

address    name
--------   ----
00001000   .bss
00000000   .data
00000080   .text
00001000   ___bss__
00000000   ___data__
00000000   ___edata__
00001006   ___end__
00000100   ___etext__
00000080   ___text__
00000000   __lflags
00000000   edata
00001006   end
00000100   etext
000000e4   hpiint
00000112   main


GLOBAL SYMBOLS: SORTED BY Symbol Address

address    name
--------   ----
00000000   ___edata__
00000000   ___data__
00000000   __lflags
00000000   edata
00000000   .data
00000080   ___text__
00000080   .text
000000e4   hpiint
00000100   etext
00000100   ___etext__
00000112   main
00001000   ___bss__
00001000   .bss
00001006   ___end__
```

```
00001006   end

[15 symbols]
```

Appendix C MELP Linker Command and Map Files

Below are example MELP linker command and map files. The linker command file contains instructions for the Texas Instruments C54xx linker utility. The map file is produced by the Texas Instruments C54xx tools after all assembling, compiling, and linking is complete; it shows the memory map and size requirements of the final executable code.

Example MELP Linker Command File

```
/*
   MELP linker command file
   Copyright (C) Signalogic, Inc. 1999-2000
*/

-c       /* Use the ROM model, init variables at runtime  */

-stack 0x400   /* stack size */
-heap  0x800   /* 1.5k heap for mallocs, etc */


MEMORY
{

        PAGE 0: RESERVED1 :     origin = 0000h length = 800h

        PAGE 0: FFT (RW):       origin = 800h length = 400h
        PAGE 0: EXT0 (RWX):     origin = 0c00h length = 100h

        PAGE 0: RESERVED2 :     origin = 0d00h length = 200h
        PAGE 0: EXT1 (RWX):     origin = 0f00h length = 100h
/*
        PAGE 0: RESERVED2 :     origin = 0d00h length = 1f0h
        PAGE 0: EXT1 (RWX):     origin = 0ef0h length = 110h
*/

        PAGE 0: DARAM (RWIX):   origin = 1000h length = 100h

        PAGE 0: EXT2 (RWX):     origin = 1200h length = 400h

        PAGE 0: EXT3 (RWX):     origin = 1600h length = 2000h

/*
        PAGE 0: EXT4 (RWX):     origin = 7c00h length = 8400h
*/
        PAGE 0: EXT4 (RWX):     origin = 7b80h length = 8480h

     /* far mode extension to code area, use 2nd 32k section of
        available ext SRAM (C549, C5409) */

        PAGE 2: EXT4 (RWX):     origin = 18000h length = 8000h


     /* Reserved3:  C54xx SCI basic stuff, add DSP BIOS if needed */

        PAGE 1: RESERVED1:      origin = 0000h length = 700h

     /* Reserved2:  talker transfer area */

        PAGE 1: RESERVED2:      origin = 1000h length = 100h
/*
        PAGE 1: EXT2 (RW):      origin = 3600h length = 3a00h
*/
```

```
     /* Bss section (heap) */

        PAGE 1: EXT2 (RW):      origin = 3600h length = 3800h

     /* Reserved3:  unused */

        PAGE 1: RESERVED3:      origin = 6e00h length = 200h

     /* Ext1, Page 1:  RTC library functions */

        PAGE 1: EXT3 (RW):      origin = 7000h length = 0c00h
}


SECTIONS {

        HI_START        100h: {.=80h; }  PAGE 0
        RESET            80h: { }        PAGE 0
        PCDATA          200h: {.=200h;}  PAGE 0
        INIT            400h: { }        PAGE 0
        MEMINI          500h: { }        PAGE 0
        VARS            42Fh: { }        PAGE 0
        DPRAM          1000h: {.=100h;}  PAGE 0

        ffttab:         > EXT0 PAGE 0
        twiddles:       > EXT2 PAGE 0

        fftdata:        > FFT PAGE 0  /* must align on 0x800 */

        isr_code:       > EXT1 PAGE 0

        .data:          > EXT1 PAGE 0
        .cinit:         > EXT3 PAGE 0
        .cd_book        > EXT3 PAGE 0
        .text:          > EXT4 PAGE 0

        far_sect:       > EXT4 PAGE 2 {

                           melp_rt.obj (.text)
                           ss_comm.obj (.text)
        }

        .lib:           > EXT3 PAGE 0

        .const:         > EXT2 PAGE 1

        .bss:           > EXT2 PAGE 1
            {
              _bssBeg = .;
              *(.bss)
              _bssSize = . - _bssBeg -1;
            }

        .stack:         > EXT2 PAGE 1
        .sysmem:        > EXT2 PAGE 1
}


/* boot / initialization, DSPower host compatibility */

t54beg
t54def
c_init

/* .c files */

melp_rt  /* includes both RT and simulation modes */
ss_comm
sd4_init
melp_ana
melp_syn
```

```
bsp
ss_init

/* Note statements in lnk.bat and lnk_far.bat that
   copy the correct .lib file to rts_x.lib prior to
   linking.  Additional notes:

   1) Placing .lib position here produces a more
      efficient link -- DO NOT add -l option to linker
      command line!
   2) Standard TI run-time near-mode library is rts.lib
   3) "" "" far-mode library is rts_ext.lib

*/

-l rts_x.lib

melp_sub
vq_lib
dsp_sub
fec_code
fs_lib
lpc_lib
mat_lib
math_lib
melp_chn
pit_lib
coeff
mathhalf
mathdp31

/* asm files */

fsvq_cbd
msvq_cbd

frm_mv
isr_sd4
/*
old: replaced by isr_sd4
tic54x
*/
isr_sync

envelope
negate
shr
shl
add
sub
fill_new
extract
deposit
zerflt
window
window_q
divide
l_sub
mult
l_mult
l_mac
v_scale
v_add
v_zap
v_sub
packcd
unpackcd
norm_s
norm_l
fpitscal
iir_2ndd
```

```
iir_2nds
lpc_syn
lpc_aejw
lpc_bwex
vequ_shr
ldivide2
lsqrtfxp
iterpary
lvinner
l_msu
l_abs
l_mpyu
vscalshl
cos_fxp
ref2pred
vq_msd2
lsp2pred
lpc_acor
lpcschur
l_shr
zerflt_q
pow10fxp
rand_num
sin_fxp
lv_magsq
l_mpy_ls
shift_r
scaladjl
vq_enc
abs_s
l_add
round
log10fxp
sqrt_fxp
peakines
lsp2freq
v_equ
cfft
findharm
vq_ms4
find_pit
idftreal

/* the following .asm files are not yet bit-exact; use C substitute */

/* l_shl */
```

Example MELP Map File

```
******************************************************************************
TMS320C54x COFF Linker           Version 3.10
******************************************************************************
>> Linked Tue Jul  9 13:41:09 2002

OUTPUT FILE NAME:   <melptest.out>
ENTRY POINT SYMBOL: "_c_int00"  address: 00008bb8


MEMORY CONFIGURATION

           name     origin    length      used    attributes    fill
         --------  --------  ---------  --------  ----------  --------
PAGE 0: RESERVED  00000000  000000800  00000549     RWIX
         FFT       00000800  000000400  00000400     RW
         EXT0      00000c00  000000100  0000001c     RW X
         RESERVED  00000d00  000000200  00000000     RWIX
         EXT1      00000f00  000000100  000000ea     RW X
         DARAM     00001000  000000100  00000100     RWIX
         EXT2      00001200  000000400  000003fe     RW X
         EXT3      00001600  000002000  00001ecf     RW X
```

```
         EXT4      00007b80  000008480  00008423     RW X

PAGE 1: RESERVED  00000000  000000700  00000000     RWIX
         RESERVED  00001000  000000100  00000000     RWIX
         EXT2      00003600  000003800  00002fa5     RW
         RESERVED  00006e00  000000200  00000000     RWIX
         EXT3      00007000  000000c00  00000000     RW


SECTION ALLOCATION MAP

 output                                  attributes/
section   page    origin      length       input sections
--------  ----  ----------  ----------   ----------------
HI_START   0    00000100    00000080     UNINITIALIZED

RESET      0    00000080    00000080
                  00000080    00000080     t54beg.obj (RESET)

PCDATA     0    00000200    00000200     UNINITIALIZED
                  00000400    00000000     t54def.obj (PCDATA)

INIT       0    00000400    00000014
                  00000400    00000014     t54beg.obj (INIT)

MEMINI     0    00000500    000000f6
                  00000500    000000f6     t54beg.obj (MEMINI)

VARS       0    0000042f    00000061
                  0000042f    00000061     t54def.obj (VARS)

DPRAM      0    00001000    00000100     UNINITIALIZED

ffttab     0    00000c00    0000001c
                  00000c00    0000001c     cfft.obj (ffttab)

twiddles   0    00001200    000003fe
                  00001200    000003fe     cfft.obj (twiddles)

fftdata    0    00000800    00000400     UNINITIALIZED
                  00000800    00000400     cfft.obj (fftdata)

isr_code   0    00000f00    000000ea
                  00000f00    0000007a     isr_sd4.obj (isr_code)
                  00000f7a    00000070     isr_sync.obj (isr_code)

.data      0    00000f00    00000000     UNINITIALIZED
                  00000f00    00000000     t54beg.obj (.data)
                  00000f00    00000000     idftreal.obj (.data)
                  00000f00    00000000     find_pit.obj (.data)
                  00000f00    00000000     vq_ms4.obj (.data)
                  00000f00    00000000     findharm.obj (.data)
                  00000f00    00000000     cfft.obj (.data)
                  00000f00    00000000     v_equ.obj (.data)
                  00000f00    00000000     lsp2freq.obj (.data)
                  00000f00    00000000     peakines.obj (.data)
                  00000f00    00000000     sqrt_fxp.obj (.data)
                  00000f00    00000000     log10fxp.obj (.data)
                  00000f00    00000000     round.obj (.data)
                  00000f00    00000000     l_add.obj (.data)
                  00000f00    00000000     abs_s.obj (.data)
                  00000f00    00000000     vq_enc.obj (.data)
                  00000f00    00000000     scaladjl.obj (.data)
                  00000f00    00000000     shift_r.obj (.data)
                  00000f00    00000000     l_mpy_ls.obj (.data)
                  00000f00    00000000     lv_magsq.obj (.data)
                  00000f00    00000000     sin_fxp.obj (.data)
                  00000f00    00000000     rand_num.obj (.data)
                  00000f00    00000000     pow10fxp.obj (.data)
                  00000f00    00000000     zerflt_q.obj (.data)
                  00000f00    00000000     l_shr.obj (.data)
```

```
00000f00    00000000     lpcschur.obj (.data)
00000f00    00000000     lpc_acor.obj (.data)
00000f00    00000000     lsp2pred.obj (.data)
00000f00    00000000     vq_msd2.obj (.data)
00000f00    00000000     ref2pred.obj (.data)
00000f00    00000000     cos_fxp.obj (.data)
00000f00    00000000     vscalshl.obj (.data)
00000f00    00000000     l_mpyu.obj (.data)
00000f00    00000000     l_abs.obj (.data)
00000f00    00000000     l_msu.obj (.data)
00000f00    00000000     lvinner.obj (.data)
00000f00    00000000     iterpary.obj (.data)
00000f00    00000000     lsqrtfxp.obj (.data)
00000f00    00000000     ldivide2.obj (.data)
00000f00    00000000     vequ_shr.obj (.data)
00000f00    00000000     lpc_bwex.obj (.data)
00000f00    00000000     lpc_aejw.obj (.data)
00000f00    00000000     lpc_syn.obj (.data)
00000f00    00000000     iir_2nds.obj (.data)
00000f00    00000000     iir_2ndd.obj (.data)
00000f00    00000000     fpitscal.obj (.data)
00000f00    00000000     norm_l.obj (.data)
00000f00    00000000     norm_s.obj (.data)
00000f00    00000000     unpackcd.obj (.data)
00000f00    00000000     packcd.obj (.data)
00000f00    00000000     v_sub.obj (.data)
00000f00    00000000     v_zap.obj (.data)
00000f00    00000000     v_add.obj (.data)
00000f00    00000000     v_scale.obj (.data)
00000f00    00000000     l_mac.obj (.data)
00000f00    00000000     l_mult.obj (.data)
00000f00    00000000     mult.obj (.data)
00000f00    00000000     l_sub.obj (.data)
00000f00    00000000     divide.obj (.data)
00000f00    00000000     window_q.obj (.data)
00000f00    00000000     window.obj (.data)
00000f00    00000000     zerflt.obj (.data)
00000f00    00000000     deposit.obj (.data)
00000f00    00000000     extract.obj (.data)
00000f00    00000000     fill_new.obj (.data)
00000f00    00000000     sub.obj (.data)
00000f00    00000000     add.obj (.data)
00000f00    00000000     shl.obj (.data)
00000f00    00000000     shr.obj (.data)
00000f00    00000000     negate.obj (.data)
00000f00    00000000     envelope.obj (.data)
00000f00    00000000     isr_sync.obj (.data)
00000f00    00000000     isr_sd4.obj (.data)
00000f00    00000000     frm_mv.obj (.data)
00000f00    00000000     msvq_cbd.obj (.data)
00000f00    00000000     fsvq_cbd.obj (.data)
00000f00    00000000     mathdp31.obj (.data)
00000f00    00000000     mathhalf.obj (.data)
00000f00    00000000     coeff.obj (.data)
00000f00    00000000     pit_lib.obj (.data)
00000f00    00000000     melp_chn.obj (.data)
00000f00    00000000     math_lib.obj (.data)
00000f00    00000000     mat_lib.obj (.data)
00000f00    00000000     lpc_lib.obj (.data)
00000f00    00000000     fs_lib.obj (.data)
00000f00    00000000     fec_code.obj (.data)
00000f00    00000000     dsp_sub.obj (.data)
00000f00    00000000     vq_lib.obj (.data)
00000f00    00000000     melp_sub.obj (.data)
00000f00    00000000     rts.lib : memmov.obj (.data)
00000f00    00000000             : memory.obj (.data)
00000f00    00000000             : exit.obj (.data)
00000f00    00000000             : boot.obj (.data)
00000f00    00000000     ss_init.obj (.data)
00000f00    00000000     bsp.obj (.data)
00000f00    00000000     melp_syn.obj (.data)
```

```
                  00000f00    00000000     melp_ana.obj (.data)
                  00000f00    00000000     sd4_init.obj (.data)
                  00000f00    00000000     melp_rt.obj (.data)
                  00000f00    00000000     c_init.obj (.data)
                  00000f00    00000000     t54def.obj (.data)

.cinit     0    00001600    000004e5
                  00001600    00000048     melp_rt.obj (.cinit)
                  00001648    00000009     melp_syn.obj (.cinit)
                  00001651    00000006     rts.lib : exit.obj (.cinit)
                  00001657    00000003             : memory.obj (.cinit)
                  0000165a    00000019     melp_sub.obj (.cinit)
                  00001673    00000004     dsp_sub.obj (.cinit)
                  00001677    00000124     fec_code.obj (.cinit)
                  0000179b    00000092     lpc_lib.obj (.cinit)
                  0000182d    00000047     melp_chn.obj (.cinit)
                  00001874    00000228     coeff.obj (.cinit)
                  00001a9c    00000011     lsqrtfxp.obj (.cinit)
                  00001aad    0000000c     lpc_acor.obj (.cinit)
                  00001ab9    00000011     pow10fxp.obj (.cinit)
                  00001aca    00000006     rand_num.obj (.cinit)
                  00001ad0    00000011     sqrt_fxp.obj (.cinit)
                  00001ae1    00000003     idftreal.obj (.cinit)
                  00001ae4    00000001     --HOLE-- [fill = 0000]

.cd_book   0    00001ae5    00001680
                  00001ae5    00000a00     fsvq_cbd.obj (.cd_book)
                  000024e5    00000c80     msvq_cbd.obj (.cd_book)

.text      0    00007b80    00008423
                  00007b80    00000000     t54beg.obj (.text)
                  00007b80    00000000     isr_sync.obj (.text)
                  00007b80    00000000     isr_sd4.obj (.text)
                  00007b80    00000000     msvq_cbd.obj (.text)
                  00007b80    00000000     fsvq_cbd.obj (.text)
                  00007b80    00000000     coeff.obj (.text)
                  00007b80    00000000     ss_init.obj (.text)
                  00007b80    00000000     bsp.obj (.text)
                  00007b80    00000000     sd4_init.obj (.text)
                  00007b80    00000000     t54def.obj (.text)
                  00007b80    00000035     c_init.obj (.text)
                  00007bb5    000003f7     melp_rt.obj (.text)
                  00007fac    000004b3     melp_ana.obj (.text)
                  0000845f    00000759     melp_syn.obj (.text)
                  00008bb8    00000046     rts.lib : boot.obj (.text)
                  00008bfe    0000003d             : exit.obj (.text)
                  00008c3b    0000024e             : memory.obj (.text)
                  00008e89    00000037             : memmov.obj (.text)
                  00008ec0    000009d1     melp_sub.obj (.text)
                  00009891    000000d6     vq_lib.obj (.text)
                  00009967    000001fe     dsp_sub.obj (.text)
                  00009b65    00000401     fec_code.obj (.text)
                  00009f66    00000160     fs_lib.obj (.text)
                  0000a0c6    0000042b     lpc_lib.obj (.text)
                  0000a4f1    000000fa     mat_lib.obj (.text)
                  0000a5eb    00000027     math_lib.obj (.text)
                  0000a612    00000349     melp_chn.obj (.text)
                  0000a95b    000007bb     pit_lib.obj (.text)
                  0000b116    0000013f     mathhalf.obj (.text)
                  0000b255    000000b7     mathdp31.obj (.text)
                  0000b30c    00000030     frm_mv.obj (.text)
                  0000b33c    0000003c     envelope.obj (.text)
                  0000b378    00000006     negate.obj (.text)
                  0000b37e    00000044     shr.obj (.text)
                  0000b3c2    00000049     shl.obj (.text)
                  0000b40b    0000000a     add.obj (.text)
                  0000b415    0000000b     sub.obj (.text)
                  0000b420    00000013     fill_new.obj (.text)
                  0000b433    0000000c     extract.obj (.text)
                  0000b43f    00000006     deposit.obj (.text)
                  0000b445    0000004d     zerflt.obj (.text)
```

```
                  0000b492    0000001c     window.obj (.text)
                  0000b4ae    0000001d     window_q.obj (.text)
                  0000b4cb    00000021     divide.obj (.text)
                  0000b4ec    0000001a     l_sub.obj (.text)
                  0000b506    0000000a     mult.obj (.text)
                  0000b510    00000010     l_mult.obj (.text)
                  0000b520    00000021     l_mac.obj (.text)
                  0000b541    0000002b     v_scale.obj (.text)
                  0000b56c    00000017     v_add.obj (.text)
                  0000b583    0000001e     v_zap.obj (.text)
                  0000b5a1    00000021     v_sub.obj (.text)
                  0000b5c2    00000062     packcd.obj (.text)
                  0000b624    0000005f     unpackcd.obj (.text)
                  0000b683    0000000f     norm_s.obj (.text)
                  0000b692    0000000b     norm_l.obj (.text)
                  0000b69d    00000084     fpitscal.obj (.text)
                  0000b721    00000095     iir_2ndd.obj (.text)
                  0000b7b6    00000078     iir_2nds.obj (.text)
                  0000b82e    00000051     lpc_syn.obj (.text)
                  0000b87f    000001ec     lpc_aejw.obj (.text)
                  0000ba6b    0000002d     lpc_bwex.obj (.text)
                  0000ba98    00000065     vequ_shr.obj (.text)
                  0000bafd    00000080     ldivide2.obj (.text)
                  0000bb7d    000002fa     lsqrtfxp.obj (.text)
                  0000be77    0000004d     iterpary.obj (.text)
                  0000bec4    00000025     lvinner.obj (.text)
                  0000bee9    00000018     l_msu.obj (.text)
                  0000bf01    00000004     l_abs.obj (.text)
                  0000bf05    00000009     l_mpyu.obj (.text)
                  0000bf0e    00000023     vscalshl.obj (.text)
                  0000bf31    000000ca     cos_fxp.obj (.text)
                  0000bffb    00000066     ref2pred.obj (.text)
                  0000c061    0000008e     vq_msd2.obj (.text)
                  0000c0ef    00000248     lsp2pred.obj (.text)
                  0000c337    00000175     lpc_acor.obj (.text)
                  0000c4ac    00000159     lpcschur.obj (.text)
                  0000c605    0000004b     l_shr.obj (.text)
                  0000c650    00000095     zerflt_q.obj (.text)
                  0000c6e5    000001b0     pow10fxp.obj (.text)
                  0000c895    00000089     rand_num.obj (.text)
                  0000c91e    000000d7     sin_fxp.obj (.text)
                  0000c9f5    00000053     lv_magsq.obj (.text)
                  0000ca48    00000015     l_mpy_ls.obj (.text)
                  0000ca5d    0000001e     shift_r.obj (.text)
                  0000ca7b    00000047     scaladjl.obj (.text)
                  0000cac2    00000078     vq_enc.obj (.text)
                  0000cb3a    0000000a     abs_s.obj (.text)
                  0000cb44    0000001b     l_add.obj (.text)
                  0000cb5f    00000011     round.obj (.text)
                  0000cb70    000001d0     log10fxp.obj (.text)
                  0000cd40    000002f5     sqrt_fxp.obj (.text)
                  0000d035    000000f9     peakines.obj (.text)
                  0000d12e    0000011c     lsp2freq.obj (.text)
                  0000d24a    0000001d     v_equ.obj (.text)
                  0000d267    0000012d     cfft.obj (.text)
                  0000d394    00000043     findharm.obj (.text)
                  0000d3d7    00000341     vq_ms4.obj (.text)
                  0000d718    00000120     find_pit.obj (.text)
                  0000d838    0000276b     idftreal.obj (.text)

.lib       0    00003165    0000036a
                  00003165    0000032c     bsp.obj (.lib)
                  00003491    0000003e     ss_init.obj (.lib)

.const     1    00003600    00000000     UNINITIALIZED

.bss       1    00003600    000023a5     UNINITIALIZED
                  00003600    00000400     fs_lib.obj (.bss)
                  00003a00    00000000     window_q.obj (.bss)
                  00003a00    00000000     window.obj (.bss)
                  00003a00    00000000     divide.obj (.bss)
```

```
00003a00    00000000     mult.obj (.bss)
00003a00    00000000     l_sub.obj (.bss)
00003a00    00000000     fill_new.obj (.bss)
00003a00    00000000     sub.obj (.bss)
00003a00    00000000     extract.obj (.bss)
00003a00    00000000     zerflt.obj (.bss)
00003a00    00000000     deposit.obj (.bss)
00003a00    00000000     l_mult.obj (.bss)
00003a00    00000000     unpackcd.obj (.bss)
00003a00    00000000     packcd.obj (.bss)
00003a00    00000000     norm_s.obj (.bss)
00003a00    00000000     fpitscal.obj (.bss)
00003a00    00000000     norm_l.obj (.bss)
00003a00    00000000     v_scale.obj (.bss)
00003a00    00000000     l_mac.obj (.bss)
00003a00    00000000     v_add.obj (.bss)
00003a00    00000000     v_sub.obj (.bss)
00003a00    00000000     v_zap.obj (.bss)
00003a00    00000000     rts.lib : boot.obj (.bss)
00003a00    00000000     ss_init.obj (.bss)
00003a00    00000000     rts.lib : memmov.obj (.bss)
00003a00    00000000     mat_lib.obj (.bss)
00003a00    00000000     vq_lib.obj (.bss)
00003a00    00000000     t54def.obj (.bss)
00003a00    00000000     t54beg.obj (.bss)
00003a00    00000000     c_init.obj (.bss)
00003a00    00000000     bsp.obj (.bss)
00003a00    00000000     sd4_init.obj (.bss)
00003a00    00000000     mathhalf.obj (.bss)
00003a00    00000000     negate.obj (.bss)
00003a00    00000000     envelope.obj (.bss)
00003a00    00000000     shr.obj (.bss)
00003a00    00000000     add.obj (.bss)
00003a00    00000000     shl.obj (.bss)
00003a00    00000000     fsvq_cbd.obj (.bss)
00003a00    00000000     mathdp31.obj (.bss)
00003a00    00000000     msvq_cbd.obj (.bss)
00003a00    00000000     isr_sd4.obj (.bss)
00003a00    00000000     frm_mv.obj (.bss)
00003a00    00000000     iir_2ndd.obj (.bss)
00003a00    00000000     scaladjl.obj (.bss)
00003a00    00000000     shift_r.obj (.bss)
00003a00    00000000     abs_s.obj (.bss)
00003a00    00000000     vq_enc.obj (.bss)
00003a00    00000000     l_mpy_ls.obj (.bss)
00003a00    00000000     zerflt_q.obj (.bss)
00003a00    00000000     l_shr.obj (.bss)
00003a00    00000000     lv_magsq.obj (.bss)
00003a00    00000000     sin_fxp.obj (.bss)
00003a00    00000000     l_add.obj (.bss)
00003a00    00000000     findharm.obj (.bss)
00003a00    00000000     cfft.obj (.bss)
00003a00    00000000     find_pit.obj (.bss)
00003a00    00000000     vq_ms4.obj (.bss)
00003a00    00000000     v_equ.obj (.bss)
00003a00    00000000     log10fxp.obj (.bss)
00003a00    00000000     round.obj (.bss)
00003a00    00000000     lsp2freq.obj (.bss)
00003a00    00000000     peakines.obj (.bss)
00003a00    00000000     lvinner.obj (.bss)
00003a00    00000000     vequ_shr.obj (.bss)
00003a00    00000000     l_msu.obj (.bss)
00003a00    00000000     l_mpyu.obj (.bss)
00003a00    00000000     l_abs.obj (.bss)
00003a00    00000000     lpc_syn.obj (.bss)
00003a00    00000000     lpc_aejw.obj (.bss)
00003a00    00000000     iir_2nds.obj (.bss)
00003a00    00000000     ldivide2.obj (.bss)
00003a00    00000000     iterpary.obj (.bss)
00003a00    00000000     lpcschur.obj (.bss)
00003a00    00000000     vq_msd2.obj (.bss)
```

```
                  00003a00    00000000     lpc_bwex.obj (.bss)
                  00003a00    00000000     lsp2pred.obj (.bss)
                  00003a00    00000000     cos_fxp.obj (.bss)
                  00003a00    00000000     vscalshl.obj (.bss)
                  00003a00    00000000     ref2pred.obj (.bss)
                  00003a00    00000309     melp_rt.obj (.bss)
                  00003d09    00000001     math_lib.obj (.bss)
                  00003d0a    00000002     dsp_sub.obj (.bss)
                  00003d0c    000008c1     idftreal.obj (.bss)
                  000045cd    0000068e     lpc_lib.obj (.bss)
                  00004c5b    00000525     melp_syn.obj (.bss)
                  00005180    00000319     melp_ana.obj (.bss)
                  00005499    0000021a     coeff.obj (.bss)
                  000056b3    00000127     fec_code.obj (.bss)
                  000057da    000000d2     lpc_acor.obj (.bss)
                  000058ac    00000077     melp_chn.obj (.bss)
                  00005923    00000022     melp_sub.obj (.bss)
                  00005945    00000022     rts.lib : exit.obj (.bss)
                  00005967    00000010     pit_lib.obj (.bss)
                  00005977    0000000d     pow10fxp.obj (.bss)
                  00005984    0000000d     lsqrtfxp.obj (.bss)
                  00005991    0000000d     sqrt_fxp.obj (.bss)
                  0000599e    00000003     isr_sync.obj (.bss)
                  000059a1    00000002     rand_num.obj (.bss)
                  000059a3    00000002     rts.lib : memory.obj (.bss)

.stack     1    000059a5    00000400     UNINITIALIZED
                  000059a5    00000000     c_init.obj (.stack)
                  000059a5    00000000     rts.lib : boot.obj (.stack)

.sysmem    1    00005da5    00000800     UNINITIALIZED
                  00005da5    00000000     rts.lib : memory.obj (.sysmem)

lib        0    000005f6    000000de
                  000005f6    000000de     sd4_init.obj (lib)


GLOBAL SYMBOLS: SORTED ALPHABETICALLY BY Name

address    name
--------   ----
00003600   .bss
00000f00   .data
00007b80   .text
00000241   ACHIST
00000242   ACLOST
00000466   ACTADD
00000237   ADCTR
00000463   AICADD
00000234   AICWRD
00000440   AMPADD
00000211   AMPSCL
0000045d   ANAADD
0000022e   ANABFK
0000025c   ANGLE
00000448   AP2ADD
00000455   APCADD
00000226   APCOEF
00000219   APSCL2
0000025b   BASANG
0000042f   BDADD
00000469   BF2ADD
0000025a   BFDONE
0000043d   BFLADD
0000043f   BFNADD
00000200   BOARD
00000456   BPCADD
00000227   BPCOEF
0000020e   BUFLEN
00000210   BUFNUM
0000024b   BUFPTR
```

```
00000431   BZYADD
00000202   BZYFLG
00008c38   C$$EXIT
00000445   CF1ADD
0000044a   CF2ADD
0000046f   CFLAG
00000217   CHANNL
00000446   CHNADD
0000024e   CHNTMP
000003c0   COEFF1
000003e0   COEFF2
00000216   COFXF1
0000021b   COFXF2
00000249   COUNT
0000020a   COUPL
00000439   CPLADD
00000243   CURCHN
00000259   CURTRC
00000238   DACTR
00000467   DCTADD
00000380   DELAY1
000003a0   DELAY2
0000025d   DENOM
00000449   DF2ADD
0000021a   DFSET2
00000441   DOFADD
00000212   DOFSET
00000261   DSIGN
00000444   DUPADD
00000215   DUPCHN
00000264   FDRVAL
00000436   FFTADD
0000d2c2   FFTBEG
00000255   FFTBUF
0000d38b   FFTEND
00000254   FFTFLG
00000207   FFTSIZ
0000023d   FILUPD
0000044d   FL1ADD
0000044e   FL2ADD
0000021e   FLLEN1
0000021f   FLLEN2
00000222   FLTYP1
0000023b   FLTYP2
00000206   FRMSIZ
0000043b   FSMADD
0000020c   FS_MODE
00000223   FS_VAL
00000451   FT1ADD
0000046a   FT2ADD
0000046c   FUPADD
0000043a   GNLADD
0000020b   GNLIST
00000275   IADD
0000026f   IA
0000026b   I
00000270   IE
00000278   IMAG
0000021d   IMAX
0000044c   IMXADD
00000f00   ISR_SD4_RX
00000f3b   ISR_SD4_TX
00000f7a   ISR_SYNC_RX
00000fac   ISR_SYNC_TX
00000274   J
00000276   LADD
00000464   LC1ADD
00000465   LC2ADD
0000044f   LG1ADD
00000450   LG2ADD
0000026c   L
```

```
00000220   LGFLG1
00000221   LGFLG2
00000235   LOGC1
00000236   LOGC2
00000272   LOGFLG
0000027b   MAGVAL
00000437   MAXADD
0000022a   MAXA
0000022b   MAXB
0000000b   MAXFFT
00000279   MAXI
00000208   MAXMAG
00000462   MDLADD
00000433   MINADD
0000027a   MINI
00000204   MINMAG
00000430   MODADD
00000233   MODULE
00000201   MODVAL
00000461   MONADD
00000232   MONTOR
0000045f   MPBADD
00000230   MPBBFK
00000258   MPBSAV
0000c000   MPIOV2
00000459   MXAADD
0000045a   MXBADD
00000272   N1
00000273   N2
0000045b   NCHADD
00000000   NOINTS
0000027c   NSTAT
0000022c   NUMCHN
0000025e   NUMER
0000023f   ONE
00000434   ORDADD
00000205   ORDVAL
00000252   OUTPC
00000246   OUTTRA
00000248   OUTTRB
00000432   OVFADD
00000203   OVFFLG
0000043e   PCBADD
0000020f   PCBFLG
00000443   PHRADD
00000214   PHZREQ
00008000   PI
00004000   PIOV2
0000024d   PRVTRG
00000453   PW1ADD
00000454   PW2ADD
00000224   PWFLG1
00000225   PWFLG2
00000273   PWRFLG
00000400   Q12CON
0000046b   QNTADD
0000023c   QNUM
00000271   QUARTN
0000025f   QUOTNT
00000277   REAL
00000256   RIB1
00000257   RIB2
0000045e   RIBADD
0000022f   RIBBFK
00000438   RIFADD
00000209   RIFLAG
0000021c   RMAX
0000044b   RMXADD
00000010   RXENBL
00000251   SAMP1
00000245   SAMPLA
```

```
00000247   SAMPLB
00000244   SAMPLE
0000045c   SBTADD
00000435   SIZADD
00000004   SPPAGE
0000024a   SPTEMP
00000452   SR2ADD
0000d2c2   STAGE1
0000d2d9   STAGE2
0000d2f0   STAGE3
0000d307   STAGE4
0000d31e   STAGE5
0000d335   STAGE6
0000d34c   STAGE7
0000d363   STAGE8
0000d37a   STAGE9
0000027d   STAT1
0000027c   STAT
0000023a   STBFLN
00000468   STMADD
00000239   STMBFK
0000022d   SUBTYP
00000253   SYNFLG
00000275   TASAV
00000276   TBSAV
00000447   TCHADD
0000024f   TEMP
00000250   TEMP2
00000260   TEMPD1
00000457   TF1ADD
00000458   TF2ADD
00000228   TFFLG1
00000229   TFFLG2
00000274   TFRFLG
00000245   TRACE1
00000247   TRACE2
0000043c   TRGADD
00000218   TRGCHN
0000024c   TRGFLG
0000020d   TRGVAL
00000020   TXENBL
00000442   WINADD
0000026e   WI
00000213   WINSCL
00000460   WNDADD
00000231   WNDBFK
0000026d   WR
00000265   XI
00000267   XL
00000269   XT
00000266   YI
00000268   YL
0000026a   YT
00000240   ZERO
00007dd2   _AudioClockAdjust
0000043f   _BFNADD
00000210   _BUFNUM
00003d09   _DEBUG
00007e7b   _GetSerialData
0000344a   _InitBSP
000032d5   _InitMcBSP
0000a483   _LSP_init
0000bf01   _L_abs
0000cb44   _L_add
0000b442   _L_deposit_h
0000b43f   _L_deposit_l
0000bafd   _L_divider2
0000cbc9   _L_log10_fxp
0000b520   _L_mac
0000b2a0   _L_mpy_ll
0000ca48   _L_mpy_ls
```

```
0000bf05   _L_mpyu
0000bee9   _L_msu
0000b510   _L_mult
0000b37c   _L_negate
0000a5eb   _L_pow_fxp
0000b1ca   _L_shift_r
0000b139   _L_shl
0000c605   _L_shr
0000bb7d   _L_sqrt_fxp
0000b4ec   _L_sub
0000a4f1   _L_v_add
0000a523   _L_v_equ
0000bec4   _L_v_inner
0000c9f5   _L_v_magsq
0000a5ca   _L_v_zap
00003a0d   _ProcIdFlag_Shifted
00003a0c   _ProcIdFlag
00003c93   _RxBP
00003c91   _RxByteFlag
00003a08   _RxDataVal
00003c95   _RxEP
00003c94   _RxMP
00003c92   _RxPTR
00000f18   _RxRead1
00000f1e   _RxRead2
00000f1e   _RxRead3
00000f7c   _RxRead4
00003c9a   _RxSyncBP
00003c9c   _RxSyncEP
00003c9b   _RxSyncMP
00003c99   _RxSyncPTR
00003ca5   _Rx_sync_dead_count
000005f6   _SD4Init
00007e01   _SendSerialData
00003491   _SyncSerialInit
00003c97   _TxBP
00003c90   _TxByteFlag
00003a07   _TxDataVal
00003c98   _TxEP
00003c96   _TxPTR
00003c9e   _TxSyncBP
00003ca0   _TxSyncEP
00003c9f   _TxSyncMP
00003c9d   _TxSyncPTR
00000f58   _TxWrite1
00000f5d   _TxWrite2
00000f63   _TxWrite3
00000fb2   _TxWrite4
00000fd1   _TxWrite5
00000400   __STACK_SIZE
00000800   __SYSMEM_SIZE
00003600   ___bss__
00001600   ___cinit__
00000f00   ___data__
00000f00   ___edata__
000059a5   ___end__
0000ffa3   ___etext__
ffffffff   ___pinit__
00007b80   ___text__
00005946   __cleanup_ptr
00000001   __lflags
00005da5   __sys_memory
0000b30c   _a_get_data
0000b324   _a_put_data
00008c38   _abort
0000cb3a   _abs_s
0000b40b   _add
00003a05   _anasyn_loopback
00008c21   _atexit
00003a04   _audio_loopback
00009e3e   _binprod_int
```

```
000055cd   _bp_cof
000054d8   _bpf_den
000054ab   _bpf_num
00009140   _bpvc_ana_init
00008ec0   _bpvc_ana
00003600   _bssBeg
000023a4   _bssSize
00003ca1   _bzyflg_rx_sync
00003c7d   _bzyflg
00003ca2   _bzyflg_tx_sync
00007b80   _c_init
00008bb8   _c_int00
00008d2b   _calloc
0000d290   _cfft
00003c2c   _chbuf_ana
00003c50   _chbuf_syn
00003a01   _complexity
0000bf31   _cos_fxp
0000929e   _dc_rmv
00003ca6   _debugvar
00005672   _disp_cof
0000b4cb   _divide_s
0000a95b   _double_chk
0000aa53   _double_ver
0000b33c   _envelope
00008bfe   _exit
0000b43b   _extract_h
0000b433   _extract_l
00003a09   _fPolledTxSync
00003a0b   _fUseSyncBit
00003a0a   _fUseSyncWord
0000b69d   _f_pitch_scale
00009b65   _fec_code
00009c86   _fec_decode
00000800   _fft_data
0000b420   _fill
0000d394   _find_harm_s1
00009f66   _find_harm
0000aab4   _find_pitch
0000d718   _find_pitch_asm
0000a09b   _findmax
0000aadf   _frac_pch
00008e06   _free
0000a0c5   _fs_init
00001ae5   _fsvq_cb
00004c5b   _fsvq_weighted
000092ea   _gain_ana
00007d8f   _getData
00003165   _getMcBSPInfo
000031f2   _getMcBSPSubReg
00003c84   _getcoeffaddr
00003c80   _getframeaddr
0000d838   _idft_real
00003d0c   _idftc
0000b721   _iir_2nd_d
0000b7b6   _iir_2nd_s
0000be77   _interp_array
0000b255   _isLwLimit
0000b27a   _isSwLimit
00004c50   _lagw_cof
00009425   _lin_int_bnd
0000cb70   _log10_fxp
00003c2a   _loopcnt
0000c337   _lpc_acor
0000b87f   _lpc_aejw
0000ba6b   _lpc_bwex
0000a0c6   _lpc_clmp
0000c0ef   _lpc_lsp2pred
0000a274   _lpc_pred2lsp
0000a37c   _lpc_pred2refl
0000bffb   _lpc_refl2pred
```

```
0000a256   _lpc_schr
0000c4ac   _lpc_schur_asm
0000b82e   _lpc_syn
000054a2   _lpf_den
00005499   _lpf_num
000045cd   _lsp_cos
00004c5a   _lsp_delta
0000d12e   _lsp_to_freq
0000b21d   _mac_r
00007bb5   _main
00008cc4   _malloc
00003cab   _mcbsp
00009967   _median
00007fac   _melp_ana
00008341   _melp_ana_init
0000a72c   _melp_chn_read
0000a612   _melp_chn_write
00007c02   _melp_real
0000845f   _melp_syn
00008a29   _melp_syn_init
00003c7e   _melpflg
00003a02   _melpmode
00008e89   _memmove
00008ca2   _minit
0000b239   _msu_r
000024e5   _msvq_cb
000058ac   _msvq_cb_mean
0000b506   _mult
0000b206   _mult_r
0000b378   _negate
00009466   _noise_est
000094c5   _noise_sup
0000b692   _norm_l
0000b683   _norm_s
00003a03   _opmode
00003cb2   _pMcBSP
0000ae7d   _p_avg_init
0000ae1c   _p_avg_update
0000b5c2   _pack_code
0000d035   _peakiness
0000b0c4   _pitch_ana_init
0000ae9f   _pitch_ana
0000575a   _pitch_dec
000056f6   _pitch_enc
000056c2   _pmat74
000056d6   _pmat84
0000c6e5   _pow10_fxp
0000952b   _q_bpvc
000095ab   _q_bpvc_dec
00009602   _q_gain
000096dc   _q_gain_dec
000099e7   _quant_u
00009a9f   _quant_u_dec
00009adb   _rand_minstdgen
0000c895   _rand_num
00003c8c   _rdcoeffptr
00003c88   _rdmemptr
00008d52   _realloc
0000cb5f   _round
00003ca4   _rx_received_sync_word
00003a00   _saturation
00009f08   _sbc_dec
00009ee0   _sbc_enc
00009f2e   _sbc_syn
0000ca7b   _scale_adj_loop
0000979a   _scale_adj
00007db1   _sendData
00003c86   _sendcoeffaddr
00003c82   _sendframeaddr
0000321b   _setMcBSPSubRegAll
00003206   _setMcBSPSubReg
```

```
0000ca5d   _shift_r3
0000ca67   _shift_r6
0000ca71   _shift_r7
0000b3c2   _shl
0000b37e   _shr
0000c91e   _sin_fxp
00003a0e   _speech_in
00003ac2   _speech_out
00003b76   _speech_save
0000cd40   _sqrt_fxp
0000b415   _sub
00003a06   _sync_serial_enable
000056ce   _syntab74
000056e6   _syntab84
00003ca3   _tx_send_sync_word
0000b624   _unpack_code
0000b56c   _v_add
0000d24a   _v_equ
0000ba98   _v_equ_shr
0000a549   _v_inner
0000a581   _v_magsq
0000bf0e   _v_scale_shl
0000b541   _v_scale
0000b5a1   _v_sub
0000b583   _v_zap
00009e64   _vgetbits
0000cac2   _vq_enc
000098f6   _vq_fsw
00009891   _vq_lspw
0000d3d7   _vq_ms4
0000c061   _vq_msd2
00009ea6   _vsetbits
00005505   _win_cof
0000b4ae   _window_Q
0000b492   _window
00003c8e   _wrcoeffptr
00003c8a   _wrmemptr
0000c650   _zerflt_Q14
0000b445   _zerflt
0000c69b   _zerflt_Q15
00001600   cinit
00000f00   edata
000059a5   end
0000ffa3   etext
ffffffff   pinit


GLOBAL SYMBOLS: SORTED BY Symbol Address

address    name
--------   ----
00000000   NOINTS
00000001   __lflags
00000004   SPPAGE
0000000b   MAXFFT
00000010   RXENBL
00000020   TXENBL
00000200   BOARD
00000201   MODVAL
00000202   BZYFLG
00000203   OVFFLG
00000204   MINMAG
00000205   ORDVAL
00000206   FRMSIZ
00000207   FFTSIZ
00000208   MAXMAG
00000209   RIFLAG
0000020a   COUPL
0000020b   GNLIST
0000020c   FS_MODE
0000020d   TRGVAL
```

```
0000020e   BUFLEN
0000020f   PCBFLG
00000210   BUFNUM
00000210   _BUFNUM
00000211   AMPSCL
00000212   DOFSET
00000213   WINSCL
00000214   PHZREQ
00000215   DUPCHN
00000216   COFXF1
00000217   CHANNL
00000218   TRGCHN
00000219   APSCL2
0000021a   DFSET2
0000021b   COFXF2
0000021c   RMAX
0000021d   IMAX
0000021e   FLLEN1
0000021f   FLLEN2
00000220   LGFLG1
00000221   LGFLG2
00000222   FLTYP1
00000223   FS_VAL
00000224   PWFLG1
00000225   PWFLG2
00000226   APCOEF
00000227   BPCOEF
00000228   TFFLG1
00000229   TFFLG2
0000022a   MAXA
0000022b   MAXB
0000022c   NUMCHN
0000022d   SUBTYP
0000022e   ANABFK
0000022f   RIBBFK
00000230   MPBBFK
00000231   WNDBFK
00000232   MONTOR
00000233   MODULE
00000234   AICWRD
00000235   LOGC1
00000236   LOGC2
00000237   ADCTR
00000238   DACTR
00000239   STMBFK
0000023a   STBFLN
0000023b   FLTYP2
0000023c   QNUM
0000023d   FILUPD
0000023f   ONE
00000240   ZERO
00000241   ACHIST
00000242   ACLOST
00000243   CURCHN
00000244   SAMPLE
00000245   TRACE1
00000245   SAMPLA
00000246   OUTTRA
00000247   TRACE2
00000247   SAMPLB
00000248   OUTTRB
00000249   COUNT
0000024a   SPTEMP
0000024b   BUFPTR
0000024c   TRGFLG
0000024d   PRVTRG
0000024e   CHNTMP
0000024f   TEMP
00000250   TEMP2
00000251   SAMP1
00000252   OUTPC
```

```
00000253   SYNFLG
00000254   FFTFLG
00000255   FFTBUF
00000256   RIB1
00000257   RIB2
00000258   MPBSAV
00000259   CURTRC
0000025a   BFDONE
0000025b   BASANG
0000025c   ANGLE
0000025d   DENOM
0000025e   NUMER
0000025f   QUOTNT
00000260   TEMPD1
00000261   DSIGN
00000264   FDRVAL
00000265   XI
00000266   YI
00000267   XL
00000268   YL
00000269   XT
0000026a   YT
0000026b   I
0000026c   L
0000026d   WR
0000026e   WI
0000026f   IA
00000270   IE
00000271   QUARTN
00000272   N1
00000272   LOGFLG
00000273   N2
00000273   PWRFLG
00000274   J
00000274   TFRFLG
00000275   TASAV
00000275   IADD
00000276   TBSAV
00000276   LADD
00000277   REAL
00000278   IMAG
00000279   MAXI
0000027a   MINI
0000027b   MAGVAL
0000027c   STAT
0000027c   NSTAT
0000027d   STAT1
00000380   DELAY1
000003a0   DELAY2
000003c0   COEFF1
000003e0   COEFF2
00000400   Q12CON
00000400   __STACK_SIZE
0000042f   BDADD
00000430   MODADD
00000431   BZYADD
00000432   OVFADD
00000433   MINADD
00000434   ORDADD
00000435   SIZADD
00000436   FFTADD
00000437   MAXADD
00000438   RIFADD
00000439   CPLADD
0000043a   GNLADD
0000043b   FSMADD
0000043c   TRGADD
0000043d   BFLADD
0000043e   PCBADD
0000043f   _BFNADD
0000043f   BFNADD
```

```
00000440   AMPADD
00000441   DOFADD
00000442   WINADD
00000443   PHRADD
00000444   DUPADD
00000445   CF1ADD
00000446   CHNADD
00000447   TCHADD
00000448   AP2ADD
00000449   DF2ADD
0000044a   CF2ADD
0000044b   RMXADD
0000044c   IMXADD
0000044d   FL1ADD
0000044e   FL2ADD
0000044f   LG1ADD
00000450   LG2ADD
00000451   FT1ADD
00000452   SR2ADD
00000453   PW1ADD
00000454   PW2ADD
00000455   APCADD
00000456   BPCADD
00000457   TF1ADD
00000458   TF2ADD
00000459   MXAADD
0000045a   MXBADD
0000045b   NCHADD
0000045c   SBTADD
0000045d   ANAADD
0000045e   RIBADD
0000045f   MPBADD
00000460   WNDADD
00000461   MONADD
00000462   MDLADD
00000463   AICADD
00000464   LC1ADD
00000465   LC2ADD
00000466   ACTADD
00000467   DCTADD
00000468   STMADD
00000469   BF2ADD
0000046a   FT2ADD
0000046b   QNTADD
0000046c   FUPADD
0000046f   CFLAG
000005f6   _SD4Init
00000800   __SYSMEM_SIZE
00000800   _fft_data
00000f00   ___data__
00000f00   .data
00000f00   edata
00000f00   ___edata__
00000f00   ISR_SD4_RX
00000f18   _RxRead1
00000f1e   _RxRead2
00000f1e   _RxRead3
00000f3b   ISR_SD4_TX
00000f58   _TxWrite1
00000f5d   _TxWrite2
00000f63   _TxWrite3
00000f7a   ISR_SYNC_RX
00000f7c   _RxRead4
00000fac   ISR_SYNC_TX
00000fb2   _TxWrite4
00000fd1   _TxWrite5
00001600   ___cinit__
00001600   cinit
00001ae5   _fsvq_cb
000023a4   _bssSize
000024e5   _msvq_cb
```

```
00003165   _getMcBSPInfo
000031f2   _getMcBSPSubReg
00003206   _setMcBSPSubReg
0000321b   _setMcBSPSubRegAll
000032d5   _InitMcBSP
0000344a   _InitBSP
00003491   _SyncSerialInit
00003600   _bssBeg
00003600   ___bss__
00003600   .bss
00003a00   _saturation
00003a01   _complexity
00003a02   _melpmode
00003a03   _opmode
00003a04   _audio_loopback
00003a05   _anasyn_loopback
00003a06   _sync_serial_enable
00003a07   _TxDataVal
00003a08   _RxDataVal
00003a09   _fPolledTxSync
00003a0a   _fUseSyncWord
00003a0b   _fUseSyncBit
00003a0c   _ProcIdFlag
00003a0d   _ProcIdFlag_Shifted
00003a0e   _speech_in
00003ac2   _speech_out
00003b76   _speech_save
00003c2a   _loopcnt
00003c2c   _chbuf_ana
00003c50   _chbuf_syn
00003c7d   _bzyflg
00003c7e   _melpflg
00003c80   _getframeaddr
00003c82   _sendframeaddr
00003c84   _getcoeffaddr
00003c86   _sendcoeffaddr
00003c88   _rdmemptr
00003c8a   _wrmemptr
00003c8c   _rdcoeffptr
00003c8e   _wrcoeffptr
00003c90   _TxByteFlag
00003c91   _RxByteFlag
00003c92   _RxPTR
00003c93   _RxBP
00003c94   _RxMP
00003c95   _RxEP
00003c96   _TxPTR
00003c97   _TxBP
00003c98   _TxEP
00003c99   _RxSyncPTR
00003c9a   _RxSyncBP
00003c9b   _RxSyncMP
00003c9c   _RxSyncEP
00003c9d   _TxSyncPTR
00003c9e   _TxSyncBP
00003c9f   _TxSyncMP
00003ca0   _TxSyncEP
00003ca1   _bzyflg_rx_sync
00003ca2   _bzyflg_tx_sync
00003ca3   _tx_send_sync_word
00003ca4   _rx_received_sync_word
00003ca5   _Rx_sync_dead_count
00003ca6   _debugvar
00003cab   _mcbsp
00003cb2   _pMcBSP
00003d09   _DEBUG
00003d0c   _idftc
00004000   PIOV2
000045cd   _lsp_cos
00004c50   _lagw_cof
00004c5a   _lsp_delta
```

```
00004c5b   _fsvq_weighted
00005499   _lpf_num
000054a2   _lpf_den
000054ab   _bpf_num
000054d8   _bpf_den
00005505   _win_cof
000055cd   _bp_cof
00005672   _disp_cof
000056c2   _pmat74
000056ce   _syntab74
000056d6   _pmat84
000056e6   _syntab84
000056f6   _pitch_enc
0000575a   _pitch_dec
000058ac   _msvq_cb_mean
00005946   __cleanup_ptr
000059a5   end
000059a5   ___end__
00005da5   __sys_memory
00007b80   _c_init
00007b80   ___text__
00007b80   .text
00007bb5   _main
00007c02   _melp_real
00007d8f   _getData
00007db1   _sendData
00007dd2   _AudioClockAdjust
00007e01   _SendSerialData
00007e7b   _GetSerialData
00007fac   _melp_ana
00008000   PI
00008341   _melp_ana_init
0000845f   _melp_syn
00008a29   _melp_syn_init
00008bb8   _c_int00
00008bfe   _exit
00008c21   _atexit
00008c38   C$$EXIT
00008c38   _abort
00008ca2   _minit
00008cc4   _malloc
00008d2b   _calloc
00008d52   _realloc
00008e06   _free
00008e89   _memmove
00008ec0   _bpvc_ana
00009140   _bpvc_ana_init
0000929e   _dc_rmv
000092ea   _gain_ana
00009425   _lin_int_bnd
00009466   _noise_est
000094c5   _noise_sup
0000952b   _q_bpvc
000095ab   _q_bpvc_dec
00009602   _q_gain
000096dc   _q_gain_dec
0000979a   _scale_adj
00009891   _vq_lspw
000098f6   _vq_fsw
00009967   _median
000099e7   _quant_u
00009a9f   _quant_u_dec
00009adb   _rand_minstdgen
00009b65   _fec_code
00009c86   _fec_decode
00009e3e   _binprod_int
00009e64   _vgetbits
00009ea6   _vsetbits
00009ee0   _sbc_enc
00009f08   _sbc_dec
00009f2e   _sbc_syn
```

```
00009f66   _find_harm
0000a09b   _findmax
0000a0c5   _fs_init
0000a0c6   _lpc_clmp
0000a256   _lpc_schr
0000a274   _lpc_pred2lsp
0000a37c   _lpc_pred2refl
0000a483   _LSP_init
0000a4f1   _L_v_add
0000a523   _L_v_equ
0000a549   _v_inner
0000a581   _v_magsq
0000a5ca   _L_v_zap
0000a5eb   _L_pow_fxp
0000a612   _melp_chn_write
0000a72c   _melp_chn_read
0000a95b   _double_chk
0000aa53   _double_ver
0000aab4   _find_pitch
0000aadf   _frac_pch
0000ae1c   _p_avg_update
0000ae7d   _p_avg_init
0000ae9f   _pitch_ana
0000b0c4   _pitch_ana_init
0000b139   _L_shl
0000b1ca   _L_shift_r
0000b206   _mult_r
0000b21d   _mac_r
0000b239   _msu_r
0000b255   _isLwLimit
0000b27a   _isSwLimit
0000b2a0   _L_mpy_ll
0000b30c   _a_get_data
0000b324   _a_put_data
0000b33c   _envelope
0000b378   _negate
0000b37c   _L_negate
0000b37e   _shr
0000b3c2   _shl
0000b40b   _add
0000b415   _sub
0000b420   _fill
0000b433   _extract_l
0000b43b   _extract_h
0000b43f   _L_deposit_l
0000b442   _L_deposit_h
0000b445   _zerflt
0000b492   _window
0000b4ae   _window_Q
0000b4cb   _divide_s
0000b4ec   _L_sub
0000b506   _mult
0000b510   _L_mult
0000b520   _L_mac
0000b541   _v_scale
0000b56c   _v_add
0000b583   _v_zap
0000b5a1   _v_sub
0000b5c2   _pack_code
0000b624   _unpack_code
0000b683   _norm_s
0000b692   _norm_l
0000b69d   _f_pitch_scale
0000b721   _iir_2nd_d
0000b7b6   _iir_2nd_s
0000b82e   _lpc_syn
0000b87f   _lpc_aejw
0000ba6b   _lpc_bwex
0000ba98   _v_equ_shr
0000bafd   _L_divider2
0000bb7d   _L_sqrt_fxp
```

```
0000be77   _interp_array
0000bec4   _L_v_inner
0000bee9   _L_msu
0000bf01   _L_abs
0000bf05   _L_mpyu
0000bf0e   _v_scale_shl
0000bf31   _cos_fxp
0000bffb   _lpc_refl2pred
0000c000   MPIOV2
0000c061   _vq_msd2
0000c0ef   _lpc_lsp2pred
0000c337   _lpc_acor
0000c4ac   _lpc_schur_asm
0000c605   _L_shr
0000c650   _zerflt_Q14
0000c69b   _zerflt_Q15
0000c6e5   _pow10_fxp
0000c895   _rand_num
0000c91e   _sin_fxp
0000c9f5   _L_v_magsq
0000ca48   _L_mpy_ls
0000ca5d   _shift_r3
0000ca67   _shift_r6
0000ca71   _shift_r7
0000ca7b   _scale_adj_loop
0000cac2   _vq_enc
0000cb3a   _abs_s
0000cb44   _L_add
0000cb5f   _round
0000cb70   _log10_fxp
0000cbc9   _L_log10_fxp
0000cd40   _sqrt_fxp
0000d035   _peakiness
0000d12e   _lsp_to_freq
0000d24a   _v_equ
0000d290   _cfft
0000d2c2   STAGE1
0000d2c2   FFTBEG
0000d2d9   STAGE2
0000d2f0   STAGE3
0000d307   STAGE4
0000d31e   STAGE5
0000d335   STAGE6
0000d34c   STAGE7
0000d363   STAGE8
0000d37a   STAGE9
0000d38b   FFTEND
0000d394   _find_harm_s1
0000d3d7   _vq_ms4
0000d718   _find_pitch_asm
0000d838   _idft_real
0000ffa3   etext
0000ffa3   ___etext__
ffffffff   pinit
ffffffff   ___pinit__

[501 symbols]
```

Appendix D C6x01 Talker Program

Below are the source code, linker command file and map file for the C6x01 talker program. See also section 5.1.5.2 above, C6x01 Source Code Interface Host PC Communication.

```
*****************************************************************************
*   Talker program for TMS320C6X01                                          *
*                                                                           *
*   COPYRIGHT (C) SIGNALOGIC, INC, 2000-2002                                *
*                                                                           *
*   Created:  Donald Eubanks, Oct. 2001                                     *
*   Modified: for C6202/3 and X-Bus support, Jeff Brower, Nov 2001          *
*                                                                           *
*****************************************************************************


FP       .set       A15
DP       .set       B14
SP       .set       B15

         .global    $bss

         .file      "C6201hi.asm"


;         .bss       _COMMAND,4
;         .bss       _PUDDADDR,4
;         .bss       _LENGTH,4
;         .bss       _DUPDADDR,4
;         .bss       _HPIbuf,4



         .sect      ".text"

         .global    _start
         .global    _TalkerInt

         .global    __STACK_SIZE
         .global    __stack

__stack: .usect     .stack, 0, 8

          .sect   "TALKVARS"
COMMAND
_COMMAND  .word      0
PUDDADDR
_PUDDADDR .word      0
LENGTH
_LENGTH   .word      0
DUPDADDR
_DUPDADDR .word      0
HPIbuf
_HPIbuf   .word      0

         .sect      "Talker"

_start:

    ; INITIALIZE Stack pointer and stack size
    MVKL            __stack,SP
    MVKH            __stack,SP
    MVKL            __STACK_SIZE - 4,B0
    MVKH            __STACK_SIZE - 4,B0
    ADD             B0,SP,SP

    ; THE SP MUST BE ALIGNED ON AN 8-BYTE BOUNDARY
    AND             ~7,SP,SP

    ; SET UP THE GLOBAL PAGE POINTER IN B14
```

```
   MVKL            $bss,DP
   MVKH            $bss,DP


   MVC     .S2     CSR,B7            ; get CSR
   AND             -2,B7,B7
   MVC     .S2     B7,CSR            ; disable all interrupts, clear GIE

   MVK     .S2     3,B8
   MVC     .S2     B8,IER            ; disable all interrupts except NMI

   MVKL    .S2     0xFFFF,B7
   MVKLH   .S2     0xFFFF,B7
   MVC     .S2     B7,ICR            ; clear all pending interrupts

   ; Initialization of EMIF global control, CE1, CE0, CE2, CE3,
   ; SDRAM Control, SDRAM Refresh, Interrupt Polarity registers
   MVKL    .S1     0x01800000,A7
   MVKL    .S2     0x00003069,B7
   MVKH    .S2     0x00003069,B7
|| MVKH    .S1     0x01800000,A7
   STW     .D1T2   B7,*A7            ; EMIF global control register
   NOP             4

   MVKL    .S1     0x01800004,A5
   MVKL    .S2     0x77E71F22,B7
   MVKH    .S2     0x77E71F22,B7
|| MVKH    .S1     0x01800004,A5
   STW     .D1T2   B7,*A5            ; CE1 Control
   NOP             4

   MVKL    .S1     0x01800008,A3
   MVKL    .S2     0x30810420,B7
   MVKH    .S2     0x30810420,B7
|| MVKH    .S1     0x01800008,A3
   STW     .D1T2   B7,*A3            ; CE0 Control
   NOP             4

   MVKL    .S1     0x01800010,A5
   MVKL    .S2     0x00000030,B7
   MVKH    .S2     0x00000030,B7
|| MVKH    .S1     0x01800010,A5
   STW     .D1T2   B7,*A5            ; CE2 Control
   NOP             4

   MVKL    .S1     0x01800014,A5
   MVKL    .S2     0x00000040,B7
   MVKH    .S2     0x00000040,B7
|| MVKH    .S1     0x01800014,A5
   STW     .D1T2   B7,*A5            ; CE3 Control
   NOP             4

   MVKL    .S2     0x07117000,B4
|| MVKL    .S1     0x01800018,A4
   MVKH    .S2     0x07117000,B4
|| MVKH    .S1     0x01800018,A4
   STW     .D1T2   B4,*A4            ; SDRAM Control
   NOP             4

   MVKL    .S1     0x0180001C,A6
   MVKL    .S2     0x00000618,B4
   MVKH    .S2     0x00000618,B4
|| MVKH    .S1     0x0180001C,A6
   STW     .D1T2   B4,*A6            ; SDRAM Refresh
   NOP             4

   MVKL    .S2     0x0000000F,B5
|| MVKL    .S1     0x019C0008,A0
   MVKH    .S2     0x0000000F,B5
|| MVKH    .S1     0x019C0008,A0
   STW     .D1T2   B5,*A0            ; Interrupt Polarity
```

```
    NOP             4
    ; END of initialization


    ; Cache initialization
    MVC     .S2     CSR,B5            ; copy control status register
 || MVKL    .S1     0xFFE3,A5
    AND     .L1X    A5,B5,A5          ; clear PCC field of CSR value
 || MVK     .S2     0x0008,B5         ; set cache enable mask
    OR      .L2X    A5,B5,B5          ; set cache enable bit
;    MVC     .S2     B5,CSR            ; update CSR to enable cache

    NOP             5
    NOP             9
    NOP             9
    NOP             5

    MVKL            0x0160FFF4,B0 ; get the check word
    MVKH            0x0160FFF4,B0
    LDW             *B0,B1
    NOP             4
 [B1] B              SetVariable
    NOP             5

    ZERO    .L1     A3
    MVKH    .S1     0x1880000,A3      ; get the address of HPIC
    LDW     .D1T1   *A3,A0
    NOP             4
    MVKL    .S1     0x00030003,A0
    MVKH    .S1     0x00030003,A0
    STW     .D1T1   A0,*A3            ; clear DSPINT bit by writing one to DSPINT
    NOP             4

  ; V360 MAILBOX INTERRUPT TALKER METHOD SETUP


    MVKL            0x01500000,A3
    MVKH            0x01500000,A3      ; address of EINT4 multiplexer
    LDW     .D1T1   *A3,A0
    NOP             4
    MVKL            0x00000008, A0
    MVKH            0x00000008, A0     ; Triggers on PCI bus (mailboxes) cause EINT4 interrupt
    STW     .D1T1   A0,*A3
    NOP             4

    MVKL            0x014000D8,A3
    MVKH            0x014000D8,A3      ; address of V3 Mailbox Interrupt Status
    LDW     .D1T1   *A3,A0
    NOP             4
    STW     .D1T1   A0,*A3             ; clear all interrupts
    NOP             5

    MVKL            0x019C0000,A3
    MVKH    .S1     0x019C0000,A3      ; address of interrupt multiplexer high
    LDW     .D1T1   *A3,A0
    NOP             4
    MVKL            0xFFE0FFFF, B6
    MVKH            0xFFE0FFFF, B6
    AND             B6, A0, A0         ; clear INTSEL13 area of IMH
    MVKL            0x00040000, B6
    MVKH            0x00040000, B6
    OR              B6, A0, A0         ; set interrupt 13 to trigger to EINT4 signal
    STW     .D1T1   A0,*A3
    NOP             4


    MVKL    .S1     0x0160FFF4,A3      ; get the check word addr
    MVKH    .S1     0x0160FFF4,A3
    MVKL    .S1     0xFAFBFCFD,A0
    MVKH    .S1     0xFAFBFCFD,A0
    STW             A0, *A3            ; set the check word
```

```
    NOP             4


SetVariable:


InterruptClear:
    MVKL    .S1     0xFFFFFFFF,A0
    MVKH    .S1     0xFFFFFFFF,A0
    MVC     .S2X    A0,ICR            ; ICR = 0xFFFF, clear all pending interrupts

    ZERO            B6
;    MVC     .S2     IER,B6            ; IER |= 0x2000, enable INT13 (DSPINT)
    MVKL    .S2     8195,B6
;    OR      .L2X    A0,B6,B6
    MVC     .S2     B6,IER


    MVC     .S2     CSR,B6            ; CSR |= 0x1, interrupts globaly enabled
    OR      .L2     1,B6,B6
    MVC     .S2     B6,CSR

Infinite:
    NOP
    B       .S1     Infinite
    NOP             5
    NOP


;*****************************************************************************
;* FUNCTION NAME: _TalkerInt                                                 *
;*                                                                           *
;*   Regs Modified     : A0,A1,A2,A3,A4,A5,A6,B0,B4,B6,SP                    *
;*   Regs Used         : A0,A1,A2,A3,A4,A5,A6,B0,B4,B6,SP                    *
;*****************************************************************************

_TalkerInt:

    ; Push to stack all used registers

    STW     .D2T2   B6,*SP--(44)
    MVC             IRP,B6
    NOP             5
    STW     .D2T2   B6,*+SP(4)              ; store IRP
    STW     .D2T1   A0,*+SP(8)
    STW     .D2T1   A1,*+SP(12)
    STW     .D2T1   A2,*+SP(16)
    STW     .D2T1   A3,*+SP(20)

    STW     .D2T1   A4,*+SP(24)
    STW     .D2T1   A5,*+SP(28)
    STW     .D2T1   A6,*+SP(32)
    STW     .D2T2   B0,*+SP(36)
    STW     .D2T2   B4,*+SP(40)

    MVKL            0x014000D8,A3
    MVKH            0x014000D8,A3      ; address of V3 Mailbox Interrupt Status
    LDW     .D1T1   *A3,A0
    NOP             4
    STW     .D1T1   A0,*A3             ; clear all interrupts
    NOP             5

    MVC             CSR, B4
    MVK             0x1,B0
    OR              B4,B0,B4   ; enable interrupts in talker
    MVC             B4,CSR

    MVKL    .S2     _COMMAND,B4
    MVKH    .S2     _COMMAND,B4
    LDW     .D2T2   *B4,B0             ; load COMMAND into B0
    NOP             4
```

```
    ; switch case statement based on (command)
    CMPEQ           B0,0,A2
    [A2] B          CASE0           ; if COMMAND=0, jump to CASE0
    NOP             5

    CMPEQ           B0,1,A2
    [A2] B          CASE1           ; if COMMAND=1, jump to CASE1
    NOP             5

    CMPEQ           B0,2,A2
    [A2] B          CASE2           ; if COMMAND=2, jump to CASE2
    NOP             5

    CMPEQ           B0,3,A2
    [A2] B          CASE3           ; if COMMAND=3, jump to CASE3
    NOP             5

    CMPEQ           B0,4,A2
    [A2] B          CASE4           ; if COMMAND=4, jump to CASE4
    NOP             5

    CMPEQ           B0,5,A2
    [A2] B          CASE5           ; if COMMAND=5, jump to CASE5
    NOP   5


CASE0:  ; if COMMAND=0 or COMMAND=3
CASE3:

    MVKL    .S2     _DUPDADDR,B4
    MVKH    .S2     _DUPDADDR,B4    ; transfer will go from A0 to A4
    NOP             3

    LDW     .D2T1   *B4,A0          ; *DUPDADDR in A0
    NOP             4


    MVKL    .S1     _PUDDADDR,A3
    MVKH    .S1     _PUDDADDR,A3
    NOP             3

    LDW     .D1T1   *A3,A4          ; *PUDDADDR in A4
    NOP             4

    B       .S1     ForLoop
    NOP             5

CASE1:  ; if COMMAND=1 or COMMAND=2
CASE2:

    MVKL    .S2     _PUDDADDR,B4
    MVKH    .S2     _PUDDADDR,B4
    NOP             3

    LDW     .D2T1   *B4,A0          ; *PUDDADDR in A0
    MVKL    .S1     _DUPDADDR,A3
    MVKH    .S1     _DUPDADDR,A3
    LDW     .D1T1   *A3,A4          ; *DUPDADDR in A4
    NOP

    NOP             5


ForLoop:
    ; FOR loop implemenation

    MVKL    .S1     _LENGTH,A3
```

```
    MVKH    .S1     _LENGTH,A3
    NOP             3

    LDW     .D1T2   *A3,B0
    NOP             4

    ADD             B0,1,B0           ; host is passing length-1, so increment length
    NOP             4

    [!B0] B .S1     RESTORE           ; if B0 is zero, go to the end (stack pop)
    NOP             5

    ZERO            A1
    NOP             4

loop1:

    CMPEQ           A1,B0,A2          ; If i = LENGTH
    NOP             4

    [A2] B          RESTORE           ; leave the loop (go to stack pop)
    NOP             5

    SHL             A1,2,A2           ; A2 = i * 4
    ADD             A4,A2,A6          ; A4* + 4*i = A6
    ADD             A0,A2,A5          ; A0* + 4*i = A5

; THIS CODE ADDED FOR DEBUG PURPOSES. IF "TALKER" VARIABLE IN TMSII6XM CODE
; IS SET TO 1, THE TALKER WILL NOT STORE ANY DATA WHATSOEVER.  DE
;    MVKL            0x80002a08, B6
;    MVKH            0x80002a08, B6

;    LDW             *B6, A2
;    NOP  4

;    [A2] B          loop2
;    NOP  5

    LDW             *A5,A2
    NOP             4

    STW             A2,*A6            ; A6* = A5*
    NOP             4

loop2:

    ADD             A1,1,A1           ; increment the counter (i=i+1)
    NOP             4

    B               loop1
    NOP             5


CASE4:  ; if COMMAND=4


    MVKL    .S1     _PUDDADDR, A1
    MVKH    .S1     _PUDDADDR, A1

    LDW    .D1T2   *A1, B0
    NOP             4


    B       .S2     B0                ; launch with no return;
    NOP             5


CASE5:

RESTORE:
```

```
    ; Pop from the stack

    LDW     .D2T2   *+SP(40),B4
    LDW     .D2T2   *+SP(36),B0
    LDW     .D2T1   *+SP(32),A6
    LDW     .D2T1   *+SP(28),A5
    LDW     .D2T1   *+SP(24),A4
    NOP             5

    LDW     .D2T1   *+SP(20),A3
    LDW     .D2T1   *+SP(16),A2

    MVKL    .S1     0x014000C8,A1
    MVKH    .S1     0x014000C8,A1
    MVKL    .S1     0x7F8F9FAF,A0
    MVKH    .S1     0x7F8F9FAF,A0
    STW     .D1T1   A0,*A1          ; set PCI mailbox register
    NOP             5

    ; Pop from the stack

    LDW     .D2T1   *+SP(12),A1
    LDW     .D2T1   *+SP(8),A0
    LDW     .D2T2   *+SP(4),B6      ; load IRP
    NOP             5
    MVC             B6,IRP
    NOP             5
    LDW     .D2T2   *++SP(44),B6
    NOP             5

    B               IRP
    NOP             5
```

Talker Command File

```
-o C6201hi.out -m C6201hi.map
-heap  0x200
-stack 0x200
/* -lrts6201.lib */


MEMORY
{

   VECS        : o = 0x00000000,  l = 0x000200   /* reset and interupt vectors */
   IPRAM       : o = 0x00000200,  l = 0x3FFE00   /* code */
   IDRAM       : o = 0x80000000,  l = 0x001000   /* .sysmem (heap), .stack */
   IDRAM2      : o = 0x80001000,  l = 0x3FF000   /* .bss */
   ASRAM       : o = 0x01600000,  l = 0x00FFFF   /* asynchronous SRAM (shared PCI) */

}

SECTIONS
{

  /* program memory sections */

        vectors  > VECS

        Talker   0x400: { } IPRAM

        .text    > IPRAM
        .const   > IPRAM
        .cio     > IPRAM
        .cinit   > IPRAM
        .debug   > IPRAM
```

```
  /* data memory sections */

        .stack   > IDRAM
        .data    > IDRAM
        .sysmem  > IDRAM
        .vars    > IDRAM
        .far     > IDRAM
        TALKVARS > ASRAM

  /* host communication area starts at 0x80001000; this
     must stay fixed to be compatible with DirectDSP driver */

        .bss     > IDRAM2

}

C6201hi
vectors6201hi
```

Talker Map File

```
******************************************************************************
          TMS320C6x COFF Linker PC Version 4.10
******************************************************************************

OUTPUT FILE NAME:   <C6201hi.out>
ENTRY POINT SYMBOL: "_start"  address: 00000400


MEMORY CONFIGURATION

                  name            origin    length      used    attr    fill
         ----------------------  --------  ---------  --------  ----  --------
         VECS                    00000000   00000200  00000200  RWIX
         IPRAM                   00000200   003ffe00  00000420  RWIX
         ASRAM                   01600000   0000ffff  00000014  RWIX
         IDRAM                   80000000   00001000  00000200  RWIX
         IDRAM2                  80001000   003ff000  00000000  RWIX


SECTION ALLOCATION MAP

 output                                  attributes/
section   page    origin      length       input sections
--------  ----  ----------  ----------   ----------------
vectors    0    00000000    00000200
                  00000000    00000200     vectors6201hi.obj (vectors)

Talker     0    00000400    00000420
                  00000400    00000420     C6201hi.obj (Talker)

IPRAM      0    00000000    00000000     UNINITIALIZED

.text      0    00000400    00000000     UNINITIALIZED
                  00000400    00000000     vectors6201hi.obj (.text)
                  00000400    00000000     C6201hi.obj (.text)

.const     0    00000200    00000000     UNINITIALIZED

.cio       0    00000200    00000000     UNINITIALIZED

.cinit     0    00000200    00000000

.debug     0    00000200    00000000     UNINITIALIZED

.stack     0    80000000    00000200     UNINITIALIZED
                  80000000    00000000     C6201hi.obj (.stack)
```

```
.data      0    80000000    00000000     UNINITIALIZED
                  80000000    00000000     C6201hi.obj (.data)
                  80000000    00000000     vectors6201hi.obj (.data)

.sysmem    0    80000000    00000000     UNINITIALIZED

.vars      0    80000000    00000000     UNINITIALIZED

.far       0    80000000    00000000     UNINITIALIZED

TALKVARS   0    01600000    00000014
                  01600000    00000014     C6201hi.obj (TALKVARS)

.bss       0    80001000    00000000     UNINITIALIZED
                  80001000    00000000     C6201hi.obj (.bss)
                  80001000    00000000     vectors6201hi.obj (.bss)


GLOBAL SYMBOLS: SORTED ALPHABETICALLY BY Name

address    name
--------   ----
80001000   $bss
80001000   .bss
80000000   .data
00000400   .text
00000610   _TalkerInt
00000200   __STACK_SIZE
80001000   ___bss__
ffffffff   ___cinit__
80000000   ___data__
80000000   ___edata__
80001000   ___end__
00000400   ___etext__
ffffffff   ___pinit__
00000400   ___text__
80000000   __stack
UNDEFED _c_int00
00000400   _start
ffffffff   cinit
80000000   edata
80001000   end
00000400   etext
ffffffff   pinit


GLOBAL SYMBOLS: SORTED BY Symbol Address

address    name
--------   ----
00000200   __STACK_SIZE
00000400   .text
00000400   _start
00000400   etext
00000400   ___text__
00000400   ___etext__
00000610   _TalkerInt
80000000   ___edata__
80000000   __stack
80000000   .data
80000000   edata
80000000   ___data__
80001000   end
80001000   .bss
80001000   ___bss__
80001000   $bss
80001000   ___end__
ffffffff   pinit
UNDEFED _c_int00
ffffffff   ___pinit__
ffffffff   cinit
```

```
ffffffff   ___cinit__

[22 symbols]
```

Appendix E C6x11 Talker Program

Below are the source code, linker command file and map file for the C6x11 talker program. See also section 5.2.5.2 above, C6x11 Source Code Interface Host PC Communication.

```
*************************************************************************
* Talker program for TMS320C6x11                                        *
*                                                                       *
*   COPYRIGHT (C) SIGNALOGIC, INC, 2000 (C6xxx)                         *
*                                                                       *
*   Rastko Selmic, Feb. 2001                                            *
*                                                                       *
*************************************************************************


FP  .set    A15
DP  .set    B14
SP  .set    B15

    .global    $bss

    .file      "talker.asm"


    .bss       _COMMAND,4
    .bss       _PUDDADDR,4
    .bss       _LENGTH,4
    .bss       _DUPDADDR,4
    .bss       _HPIbuf,4


    .sect      ".text"

    .global    _start
    .global    _TalkerInt

    .global    __STACK_SIZE
    .global    __stack
__stack: .usect   .stack, 0, 8


    .sect      "Talker"


_start:
    ; INITIALIZE Stack pointer and stack size
    MVKL            __stack,SP
    MVKH            __stack,SP
    MVKL            __STACK_SIZE - 4,B0
    MVKH            __STACK_SIZE - 4,B0
    ADD             B0,SP,SP

    ; THE SP MUST BE ALIGNED ON AN 8-BYTE BOUNDARY
    AND             ~7,SP,SP

    ; SET UP THE GLOBAL PAGE POINTER IN B14
    MVKL            $bss,DP
    MVKH            $bss,DP


    MVC     .S2     CSR,B7            ; get CSR
    AND             -2,B7,B7
    MVC     .S2     B7,CSR            ; disable all interrupts, clear GIE

    MVK     .S2     3,B8
    MVC     .S2     B8,IER            ; disable all interrupts except NMI

    MVKL    .S2     0xFFFF,B7
    MVKH    .S2     0xFFFF,B7
```

```
   MVC     .S2     B7,ICR            ; clear all pending interrupts

   ; Initialization of EMIF global control, EMIF CE0control, EMIF_CE1,
   ; EMIF SDRAM control, EMIF_SDRP, EMIF_SDEXT
   ZERO    .L1     A7
   MVKL    .S2     0x00003040,B7
   MVKH    .S2     0x00003040,B7
|| MVKH    .S1     0x1800000,A7
   STW     .D1T2   B7,*A7            ; EMIF global control register
   NOP             4

   MVKL    .S1     0x1800004,A5
   MVKL    .S2     0xFFFFFF23,B7
   MVKH    .S2     0xFFFFFF23,B7
|| MVKH    .S1     0x1800004,A5
   STW     .D1T2   B7,*A5            ; CE1 - 32-bit asynch access after boot
   NOP             4

   MVKL    .S1     0x1800008,A3
   MVKL    .S2     0xFFFFFF30,B7
   MVKH    .S2     0xFFFFFF30,B7
|| MVKH    .S1     0x1800008,A3
   STW     .D1T2   B7,*A3            ; CE0 - SDRAM
   NOP             4

   MVKL    .S1     0x1800010,A5
   MVKL    .S2     0xFFFFFF23,B7
   MVKH    .S2     0xFFFFFF23,B7
|| MVKH    .S1     0x1800010,A5
   STW     .D1T2   B7,*A5            ; CE2 - 32-bit asynch on daughterboard
   NOP             4

   MVKL    .S1     0x1800014,A5
   MVKL    .S2     0xFFFFFF23,B7
   MVKH    .S2     0xFFFFFF23,B7
|| MVKH    .S1     0x1800014,A5
   STW     .D1T2   B7,*A5            ; CE3 - 32-bit asynch on daughterboard
   NOP             4

   MVKL    .S2     0x07117000,B4
|| MVKL    .S1     0x1800018,A4
   MVKH    .S2     0x07117000,B4
|| MVKH    .S1     0x1800018,A4
   STW     .D1T2   B4,*A4            ; SDRAM control register (100 MHz)
   NOP             4

   MVKL    .S1     0x180001C,A6
   MVKL    .S2     0x0000061A,B4
   MVKH    .S2     0x0000061A,B4
|| MVKH    .S1     0x180001C,A6
   STW     .D1T2   B4,*A6            ; SDRAM Timing register
   NOP             4

   MVKL    .S2     0x00054519,B5
|| MVKL    .S1     0x1800020,A0
   MVKH    .S2     0x00054519,B5
|| MVKH    .S1     0x1800020,A0
   STW     .D1T2   B5,*A0            ; SDRAM Extension register
   NOP             4
   ; END of initialization


   ; Cache initialization
   MVC     .S2     CSR,B5            ; copy control status register
|| MVKL    .S1     0xFF1F,A5
   AND     .L1X    A5,B5,A5          ; clear PCC field of CSR value
|| MVK     .S2     0x0000,B5         ; set cache enable mask
   OR      .L2X    A5,B5,B5          ; set cache enable bit
   MVC     .S2     B5,CSR            ; update CSR to enable cache
   NOP             4
   NOP
```

```
    ZERO    .L1     A3
    MVKH    .S1     0x1880000,A3      ; get the address of HPIC
    LDW     .D1T1   *A3,A0
    NOP             4
    MVKL    .S1     0x00020002,A0
    MVKH    .S1     0x00020002,A0
    STW     .D1T1   A0,*A3            ; clear DSPINT bit by writing one to DSPINT
    NOP             4

    MVKL    .S1     0xFFFFFFFF,A0
    MVKH    .S1     0xFFFFFFFF,A0
    MVC     .S2X    A0,ICR            ; ICR = 0xFFFF, clear all pending interrupts

    MVC     .S2     IER,B6            ; IER |= 0x2000, enable INT13 (DSPINT)
 || MVKL    .S1     8192,A0
    OR      .L2X    A0,B6,B6
    MVC     .S2     B6,IER

    MVC     .S2     CSR,B6            ; CSR |= 0x1, interrupts globaly enabled
    OR      .L2     1,B6,B6
    MVC     .S2     B6,CSR

Infinite:
    NOP
    B       .S1     Infinite
    NOP             5
    NOP


;*************************************************************************
;* FUNCTION NAME: _TalkerInt                                             *
;*                                                                       *
;*   Regs Modified     : A0,A1,A2,A3,A4,A5,A6,B0,B4,B6,SP                *
;*   Regs Used         : A0,A1,A2,A3,A4,A5,A6,B0,B4,B6,SP                *
;*************************************************************************

_TalkerInt:

    ; Push to stack all used registers
    STW     .D2T2   B6,*SP--(40)
    STW     .D2T1   A0,*+SP(4)
    STW     .D2T1   A1,*+SP(8)
    STW     .D2T1   A2,*+SP(12)
    STW     .D2T1   A3,*+SP(16)
    STW     .D2T1   A4,*+SP(20)
    STW     .D2T1   A5,*+SP(24)
    STW     .D2T1   A6,*+SP(28)
    STW     .D2T2   B0,*+SP(32)
    STW     .D2T2   B4,*+SP(36)
    NOP             4


    ; clear DSPINT bit inside HPIC and thus enable new interrupts from host
    ZERO    .L1     A1
    MVKH    .S1     0x1880000,A1      ; address of HPIC
    LDW     .D1T1   *A1,A0
    NOP             4
    MVKL    .S1     0x00020002,A0
    MVKH    .S1     0x00020002,A0     ; setting the bits to HPIC
    STW     .D1T1   A0,*A1            ; writing to HPIC
    NOP             4


    MVKL    .S2     _COMMAND,B4
    MVKH    .S2     _COMMAND,B4
    LDW     .D2T2   *B4,B0            ; load COMMAND into B0
    NOP             4

    ; switch case statement based on (command)
```

```
    CMPEQ           B0,0,A2
    [A2] B          CASE0           ; if COMMAND=0, jump to CASE0
    NOP             5
    CMPEQ           B0,1,A2
    [A2] B          CASE1           ; if COMMAND=1, jump to CASE1
    NOP             5
    CMPEQ           B0,2,A2
    [A2] B          CASE2           ; if COMMAND=2, jump to CASE2
    NOP             5
    CMPEQ           B0,3,A2
    [A2] B          CASE3           ; if COMMAND=3, jump to CASE3
    NOP             5
    CMPEQ           B0,4,A2
    [A2] B          CASE4           ; if COMMAND=4, jump to CASE4
    NOP             5
    CMPEQ           B0,5,A2
    [A2] B          CASE5           ; if COMMAND=5, jump to CASE5
    NOP   5


CASE0:  ; if COMMAND=0 or COMMAND=3
CASE3:
    MVKL    .S2     _DUPDADDR,B4
    MVKH    .S2     _DUPDADDR,B4    ; transfer will go from A0 to A4
    LDW     .D2T1   *B4,A0          ; *DUPDADDR in A0
    NOP             4
    MVKL    .S1     _PUDDADDR,A3
    MVKH    .S1     _PUDDADDR,A3
    LDW     .D1T1   *A3,A4          ; *PUDDADDR in A4
    NOP             4
    B       .S1     ForLoop
    NOP             5

CASE1:  ; if COMMAND=1 or COMMAND=2
CASE2:
    MVKL    .S2     _PUDDADDR,B4
    MVKH    .S2     _PUDDADDR,B4
    NOP             4
    LDW     .D2T1   *B4,A0          ; *PUDDADDR in A0
    MVKL    .S1     _DUPDADDR,A3
    MVKH    .S1     _DUPDADDR,A3
    LDW     .D1T1   *A3,A4          ; *DUPDADDR in A4
    NOP             4

ForLoop:
    ; FOR loop implemenation
    MVKL    .S1     _LENGTH,A3
    MVKH    .S1     _LENGTH,A3
    LDW     .D1T2   *A3,B0
    NOP             4
    ADD             B0,1,B0         ; host is passing length-1, so increment length

    [!B0] B .S1     RESTORE         ; if B0 is zero, go to the end (stack pop)
    NOP             5

    ZERO            A1
loop1:
    CMPEQ           A1,B0,A2        ; If i = LENGTH
    [A2] B          RESTORE         ; leave the loop (go to stack pop)
    NOP             5

    SHL             A1,2,A2         ; A2 = i * 4
    ADD             A4,A2,A6        ; A4* + 4*i = A6
    ADD             A0,A2,A5        ; A0* + 4*i = A5

    LDW             *A5,A2
    NOP             4
    STW             A2,*A6          ; A6* = A5*
    NOP             4
```

```
    ADD             A1,1,A1           ; increment the counter (i=i+1)
    B               loop1
    NOP             5


CASE4:  ; if COMMAND=4
    MVC     .S2     CSR,B6            ; get CSR
    AND             -2,B6,B6
    MVC     .S2     B6,CSR            ; disable all interrupts, clear GIE

    MVKL    .S1     _PUDDADDR, A1
    MVKH    .S1     _PUDDADDR, A1
    LDW     .D1T2   *A1, B0
    NOP             4

    B       .S2     B0                ; launch with no return;
    NOP             5


CASE5:

RESTORE:

    ; Pop from the stack
    LDW     .D2T2   *+SP(36),B4
    LDW     .D2T2   *+SP(32),B0
    LDW     .D2T1   *+SP(28),A6
    LDW     .D2T1   *+SP(24),A5
    LDW     .D2T1   *+SP(20),A4
    NOP             4

    ; Signal to HOST that it is done, setting HINT bit
    ZERO    .L1     A3
    MVKH    .S1     0x1880000,A3
    LDW     .D1T1   *A3,A0
    NOP             4
    MVKL    .S1     0x00040004,A0
    MVKH    .S1     0x00040004,A0
    STW     .D1T1   A0,*A3            ; set HINT bit in HPIC
    NOP             4
    NOP

    ; Pop from the stack
    LDW     .D2T1   *+SP(16),A3
    LDW     .D2T1   *+SP(12),A2
    LDW     .D2T1   *+SP(8),A1
    LDW     .D2T1   *+SP(4),A0
 || B       .S2     IRP
    LDW     .D2T2   *++SP(40),B6
    NOP             4
```

Talker Command File

```
-o DSKC6XHI.OUT -m DSKC6XHI.MAP
-heap  0x200
-stack 0x200
/* -lrts6201.lib */


MEMORY
{

   VECS        : o = 0x00000000,  l = 0x000200   /* reset and interupt vectors */
   IPRAM1      : o = 0x00000200,  l = 0x000800   /* intended for initialization */
   IPRAM2      : o = 0x00001000,  l = 0x000500   /* intended for initialization */
   IDRAM       : o = 0x80000000,  l = 0xFFFFFF   /* .bss, .sysmem, .stack, .cinit */

}
```

```
SECTIONS
{
        vectors  > VECS
        Talker   0x400: { } IPRAM1
        .bss     > IPRAM2
        .cinit   > IPRAM1
        .text    > IPRAM1
        .stack   > IPRAM1
        .data    > IPRAM1
        .sysmem  > IPRAM1
        .const   > IPRAM1
        .cio     > IPRAM1
        .debug   > IPRAM1
        .vars    > IPRAM1
        .far     > IPRAM1

}
```

Talker Map File

```
******************************************************************************
          TMS320C6x COFF Linker PC Version 4.10
******************************************************************************
>> Linked Wed Jul 31 14:19:25 2002

OUTPUT FILE NAME:   <DSKC6XHI.OUT>
ENTRY POINT SYMBOL: "_c_int00"  address: 00000000


MEMORY CONFIGURATION

                  name            origin    length      used    attr    fill
         ----------------------  --------  ---------  --------  ----  --------
         VECS                    00000000   00000200  00000200  RWIX
         IPRAM1                  00000200   00000800  00000540  RWIX
         IPRAM2                  00001000   00000500  00000014  RWIX
         IDRAM                   80000000   00ffffff  00000000  RWIX


SECTION ALLOCATION MAP

 output                                  attributes/
section   page    origin      length       input sections
--------  ----  ----------  ----------   ----------------
vectors    0    00000000    00000200
                  00000000    00000200     vectors6xhi.obj (vectors)

Talker     0    00000400    00000340
                  00000400    00000340     talker.obj (Talker)

IPRAM1     0    00000000    00000000     UNINITIALIZED

.bss       0    00001000    00000014     UNINITIALIZED
                  00001000    00000014     talker.obj (.bss)
                  00001014    00000000     vectors6xhi.obj (.bss)

.cinit     0    00000200    00000000

.text      0    00000400    00000000     UNINITIALIZED
                  00000400    00000000     vectors6xhi.obj (.text)
                  00000400    00000000     talker.obj (.text)

.stack     0    00000200    00000200     UNINITIALIZED
                  00000200    00000000     talker.obj (.stack)

.data      0    00000200    00000000     UNINITIALIZED
                  00000200    00000000     talker.obj (.data)
```

```
                  00000200    00000000     vectors6xhi.obj (.data)

.sysmem    0    00000200    00000000     UNINITIALIZED

.const     0    00000200    00000000     UNINITIALIZED

.cio       0    00000200    00000000     UNINITIALIZED

.debug     0    00000200    00000000     UNINITIALIZED

.vars      0    00000200    00000000     UNINITIALIZED

.far       0    00000200    00000000     UNINITIALIZED


GLOBAL SYMBOLS: SORTED ALPHABETICALLY BY Name

address    name
--------   ----
00001000   $bss
00001000   .bss
00000200   .data
00000400   .text
00000578   _TalkerInt
00000200   __STACK_SIZE
00001000   ___bss__
ffffffff   ___cinit__
00000200   ___data__
00000200   ___edata__
00001014   ___end__
00000400   ___etext__
ffffffff   ___pinit__
00000400   ___text__
00000200   __stack
UNDEFED _c_int00
00000400   _start
ffffffff   cinit
00000200   edata
00001014   end
00000400   etext
ffffffff   pinit


GLOBAL SYMBOLS: SORTED BY Symbol Address

address    name
--------   ----
00000200   __STACK_SIZE
00000200   edata
00000200   ___data__
00000200   ___edata__
00000200   __stack
00000200   .data
00000400   _start
00000400   ___text__
00000400   ___etext__
00000400   .text
00000400   etext
00000578   _TalkerInt
00001000   .bss
00001000   $bss
00001000   ___bss__
00001014   end
00001014   ___end__
ffffffff   ___cinit__
ffffffff   pinit
UNDEFED _c_int00
ffffffff   cinit
ffffffff   ___pinit__

[22 symbols]
```